# IMPLEMENTATION PLAN

****Proje:** AuraStream v3.4 (Enterprise Grade)**

****Mimari:** Next.js 14 Monorepo, Supabase (Normalized), AWS SQS/Lambda, Web Audio API**

CRITICAL ARCHITECTURE DECISIONS
**ORM Yasağı (No-Prisma Rule):**

- **Karar:** Projede Prisma veya Drizzle gibi ORM araçları **KESİNLİKLE KULLANILMAYACAKTIR.**
- **Gerekçe:** Next.js Serverless ortamında "Cold Start" sürelerini artırması ve Supabase RLS (Row Level Security) politikalarını bypass etme riski taşıması.
- **Standart:** Veritabanı erişimi sadece **Native Supabase Client (`@supabase/ssr`)** ile yapılacak ve TypeScript tipleri `supabase gen types` komutu ile otomatik üretilecektir.

## 1. PROJECT STRUCTURE (KLASÖR YAPISI)

**Monorepo yapısı korunmuş, backend servisleri SQS mimarisine göre güncellenmiştir.***

**/src**

**├── app/**

**│ ├── (auth)/ # Login, Signup**

**│ ├── (venue)/ # B2B App (Player, Schedule, Legal) Next.js PWA**

**│ ├── (creator)/ # B2C App (Store, Library, Dispute Center) Next.js**

**│ ├── (admin)/ # Admin (QC, Upload, Analytics) Next.js**

**│ └── api/ # Backend Routes**

**│ ├── queue/ # SQS Job Triggers (Yeni)**

```
│   ├── webhooks/          # Suno/Stripe Callbacks
│   └── stream/            # Signed URL Generator
│   ├── search/            # Meilisearch Proxy & Analytics
│   └── health/            # System Status Check (Uptime Robot için)
├── components/
│   ├── player/            # Audio Engine UI
│   ├── venue/             # Venue Components
│   └── shared/            # Common UI
├── /packages
│   ├── /audio-engine (Web Audio API wrapper)
│   ├── /offline-manager (IndexedDB + Encryption)
│   ├── /search-client (Meilisearch + Supabase sync)
│   └── /ui-kit (Shared Tailwind components)
├── lib/
│   ├── audio/             # SES MOTORU
│   │   ├── Engine.ts      # Web Audio API (Gapless/Crossfade Logic)
│   │   └── Frequency.ts   # 432Hz DSP Logic
│   ├── queue/             # AWS SQS Client (Yeni)
│   ├── services/          # S3, Suno, PDF Service
│   └── logic/             # OfflineManager (IndexedDB)
│   ├── search/            # ARAMA MOTORU (YENİ)
│   │   ├── client.ts      # Meilisearch Config
│   │   └── sync.ts        # Supabase -> Meilisearch Sync Logic
│   ├── monitoring/        # GÖZLEMLEME (YENİ)
│   │   ├── sentry.ts      # Error Tracking
```

```
│   │   └── logger.ts            # Structured Logging (Pino/Winston)
│   └── ... (audio, queue, services mevcut yapısı)
│
└── tests/                       # TEST SUITE (YENİ)
    ├── unit/                    # Jest Tests
    │   └── e2e/                 # Playwright Tests
    │
    └── types/                   # DB Types
    ├── /context-docs     ← YENİ: Her modül için AI bağlam dosyaları
    ├── /prompt-templates ← YENİ: Görev bazlı prompt kütüphanesi
    └── /validation-scripts ← YENİ: Her modülü test eden basit
script'ler
```

BÖLÜM 1.2: ENVIRONMENT VARIABLES MASTER LIST

## 1.2. Environment Variables Master List

### Local Development (.env.local)

==========

# SUPABASE CONFIGURATION

==========

NEXT_PUBLIC_SUPABASE_URL=https://[project-id].supabase.co
NEXT_PUBLIC_SUPABASE_ANON_KEY=[anon-key-here]
SUPABASE_SERVICE_ROLE_KEY=[service-role-key-here] # BACKEND ONLY
SUPABASE_DB_URL=postgresql://postgres:[password]@db.[project-id].supabase.co:5432/postgres

==========

# SEARCH ENGINE

==========

MEILISEARCH_HOST=http://localhost:7700 # Docker için
MEILISEARCH_API_KEY=masterKey # Development key
NEXT_PUBLIC_MEILISEARCH_HOST=http://localhost:3000/api/search-proxy
MEILISEARCH_SYNC_SECRET=[random-32-char-string] # Webhook güvenliği için

==========

# AWS SERVICES

==========

AWS_ACCESS_KEY_ID=[your-access-key]
AWS_SECRET_ACCESS_KEY=[your-secret-key]
AWS_REGION=eu-central-1
AWS_S3_BUCKET_RAW=aurastream-raw-dev
AWS_S3_BUCKET_PROCESSED=aurastream-processed-dev
AWS_S3_BUCKET_PUBLIC=aurastream-public-dev
AWS_CLOUDFRONT_DISTRIBUTION_ID=[distribution-id]
AWS_CLOUDFRONT_DOMAIN=d123456abcdefg.cloudfront.net
AWS_SQS_QUEUE_URL=https://sqs.eu-central-1.amazonaws.com/[account-id]/audio-processing-queue
AWS_SQS_DLQ_URL=https://sqs.eu-central-1.amazonaws.com/[account-id]/audio-processing-dlq

==========

# PAYMENT PROCESSING

==========

STRIPE_SECRET_KEY=sk_test_51xxx
STRIPE_WEBHOOK_SECRET=whsec_xxx
NEXT_PUBLIC_STRIPE_PUBLISHABLE_KEY=pk_test_51xxx
STRIPE_PRICE_ID_BASIC=price_xxx
STRIPE_PRICE_ID_PRO=price_xxx
STRIPE_PRICE_ID_BUSINESS=price_xxx

==========

# AI GENERATION (SUNO)

==========

SUNO_API_KEY=[your-suno-api-key]
SUNO_API_ENDPOINT=https://api.suno.ai/v1
SUNO_API_RATE_LIMIT=10 # requests per minute
SUNO_COST_PER_TRACK=0.50 # USD

==========

# MONITORING & ANALYTICS

==========

SENTRY_DSN=https://[key]@o450689.ingest.sentry.io/450689
NEXT_PUBLIC_SENTRY_DSN=https://
[key]@o450689.ingest.sentry.io/450689

VERCEL_ANALYTICS_ID=prj_xxx
UPTIME_ROBOT_API_KEY=[your-key]

==========

# APPLICATION CONFIG

==========

NEXTAUTH_SECRET=[32-char-random-string]
NEXTAUTH_URL=http://localhost:3000
ENCRYPTION_KEY=[32-char-key-for-client-side-encryption]
SESSION_TIMEOUT=28800 # 8 hours in seconds
MAX_CACHE_SIZE_MB=500
DEFAULT_CROSSFADE_DURATION=3 # seconds

==========

# FEATURE FLAGS

==========

NEXT_PUBLIC_ENABLE_432HZ=true
NEXT_PUBLIC_ENABLE_OFFLINE_MODE=true
NEXT_PUBLIC_ENABLE_EXPERIMENTAL_SEARCH=false

### Production (.env.production) Farkları:

# Sadece production'da farklı olacaklar

MEILISEARCH_HOST=https://meilisearch.prod.aurastream.com
MEILISEARCH_API_KEY=[production-api-key]

AWS_S3_BUCKET_RAW=aurastream-raw-prod
AWS_S3_BUCKET_PROCESSED=aurastream-processed-prod
STRIPE_SECRET_KEY=sk_live_xxx
NEXTAUTH_URL=https://app.aurastream.com

### Environment Variables Validation Script:

```bash
#!/bin/bash
# scripts/validate-env.sh

required_vars=(
  "NEXT_PUBLIC_SUPABASE_URL"
  "NEXT_PUBLIC_SUPABASE_ANON_KEY"
  "SUPABASE_SERVICE_ROLE_KEY"
  "MEILISEARCH_HOST"
  "AWS_ACCESS_KEY_ID"
  "STRIPE_SECRET_KEY"
)

for var in "${required_vars[@]}"; do
  if [ -z "${!var}" ]; then
    echo "❌ Missing required environment variable: $var"
    exit 1
  fi
done

echo "✅ All required environment variables are set"
```

### **BÖLÜM 1.3: BUILD ORDER & DEPENDENCY MATRIX**

## 1.3. Build Order & Dependency Matrix

### Phase 1: Foundation (Week 1-2)

### BÖLÜM 1.3: BUILD ORDER & DEPENDENCY MATRIX**

## 1.3. Build Order & Dependency Matrix

### Phase 1: Foundation (Week 1-2)

packages/ui-kit
├── Status: MUST BUILD FIRST
├── Dependencies: None
├── Output: Button, Input, Card, Modal components
└── Test: Storybook + Jest
packages/database-client
├── Dependencies: None
├── Output: Supabase client wrapper with TypeScript types
└── Includes: RLS helper functions, realtime subscriptions
lib/services/s3
├── Dependencies: None
├── Output: AWS S3 client with signed URL generation
└── Features: Upload, download, delete, presigned URLs
packages/search-client
├── Dependencies: database-client
├── Output: Meilisearch client + Supabase sync logic
└── Features: Real-time sync, error recovery, health checks

### Phase 2: Core Services (Week 3-4)

lib/audio/engine
├── Dependencies: None (pure Web Audio API)

├── Output: Audio playback, crossfade, 432Hz transposition
└── Test: Mock AudioContext, offline rendering
lib/queue
├── Dependencies: None
├── Output: AWS SQS client with retry logic
└── Features: Message processing, DLQ handling, backoff
packages/offline-manager
├── Dependencies: lib/audio/engine, lib/services/s3
├── Output: IndexedDB wrapper with encryption
└── Features: Cache management, background sync, quota

### Phase 3: Applications (Week 5–8)

apps/creator-store
├── Dependencies: ui-kit, search-client, database-client
├── Entry point: /apps/creator-store/app/page.tsx
└── Features: Search, licensing, library management
apps/venue-player
├── Dependencies: ui-kit, audio-engine, offline-manager
├── Entry point: /apps/venue-player/app/page.tsx
└── Features: Player, schedule, offline cache
apps/admin-factory
├── Dependencies: ui-kit, search-client, queue
├── Entry point: /apps/admin-factory/app/page.tsx
└── Features: QC, AI generation, analytics

### Dependency Graph Visualization:
```mermaid
```

```
graph TD
    A[ui-kit] --> B[database-client]
    A --> C[search-client]
    B --> C
    A --> D[venue-player]
    E[audio-engine] --> D
    F[offline-manager] --> D
    A --> G[creator-store]
    C --> G
    B --> G
    A --> H[admin-factory]
    C --> H
    I[queue] --> H
```

## Build Scripts (package.json):

```
json
{
  "scripts": {
    "dev": "turbo dev --parallel",
    "build": "turbo build",
    "build:ci": "turbo build --filter=./packages/* --
filter=./apps/*",
    "test": "turbo test",
    "lint": "turbo lint",
    "type-check": "turbo type-check",
    "db:reset": "cd packages/database-client && tsx
scripts/reset-db.ts",
    "db:seed": "cd packages/database-client && tsx scripts/
seed-data.ts",
    "search:init": "cd packages/search-client && tsx
scripts/init-meilisearch.ts",
    "search:sync": "cd packages/search-client && tsx
scripts/full-sync.ts",
    "audio:test": "cd lib/audio/engine && npm run
test:integration",
    "docker:up": "docker-compose -f docker-compose.dev.yml
up -d",
    "docker:down": "docker-compose -f docker-
compose.dev.yml down"
  }
```

```
}


### **YENİ BÖLÜM: 1.4 API ENDPOINT SPECIFICATIONS**

## 1.4. API Endpoint Specifications

### Search API

```typescript
/**
 * @route GET /api/tracks/search
 * @description Search tracks with filters
 * @access Public (rate limited)
 */
interface SearchRequest {
  q?: string;                // Search query
  filters?: string;          // JSON stringified filters
  page?: number;             // Default: 1
  limit?: number;            // Default: 20, Max: 100
  sortBy?: 'relevance' | 'newest' | 'popular';
}

interface SearchResponse {
  tracks: Track[];
  total: number;
  page: number;
  totalPages: number;
  facets: {
    bpm: { min: number; max: number };
    duration: { min: number; max: number };
    genres: { name: string; count: number }[];
    moods: { name: string; count: number }[];
  };
}

// Error Responses:
// 400 - Invalid filters format
// 429 - Rate limit exceeded (20 req/min for IP)
```

# Stream URL Generation

```
typescript
```
```

```
/**
 * @route GET /api/stream/:trackId/:fileType
 * @description Generate signed URL for audio stream
 * @access Authenticated (JWT required)
 */
interface StreamRequest {
  trackId: string;           // UUID format
  fileType: 'aac' | 'flac';  // Audio format
  tuning?: '440' | '432';    // Optional frequency
}

interface StreamResponse {
  url: string;               // Signed CloudFront URL
  expiresAt: string;         // ISO timestamp
  contentType: string;       // audio/aac or audio/flac
  contentLength: number;     // Bytes
}

// Error Responses:
// 401 - Not authenticated
// 403 - No subscription access
// 404 - Track not found
// 429 - Rate limit (1000 req/min for venues)
```

## License Generation

```
typescript
/**
 * @route POST /api/licenses/generate
 * @description Generate license for track purchase
 * @access Authenticated (JWT required)
 */
interface LicenseRequest {
  trackId: string;
  projectName: string;
  usageType: 'youtube' | 'podcast' | 'advertisement';
  formats: ('mp3' | 'flac' | 'wav')[];
  frequencies: ('440' | '432')[];
  youtubeChannelId?: string;
  agreeToTerms: boolean;
  agreeToLiability: boolean;
}
```

```
interface LicenseResponse {
  licenseId: string;
  downloadUrl: string;        // 7-day expiry
  pdfUrl: string;            // License certificate
  assetId: string;           // For YouTube disputes
  createdAt: string;
  expiresAt: string;         // License validity
}

// Error Responses:
// 400 - Missing required fields
// 402 - Payment required
// 409 - Already licensed for same project
```

**Custom Request API**

- `POST /api/requests/create` (Sipariş oluşturma)
- `GET /api/requests/my-orders` (Sipariş takibi)

# AI Generation Webhook

```
typescript
/**
 * @route POST /api/webhooks/suno
 * @description Receive generated audio from Suno AI
 * @access Suno AI (API key validation)
 */
interface SunoWebhook {
  id: string;
  status: 'completed' | 'failed';
  audio_url?: string;
  metadata?: {
    title?: string;
    duration?: number;
    bpm?: number;
    instruments?: string[];
  };
  error?: string;
}
```

```
// Response: 200 OK (always return 200 to acknowledge)
// Security: Validate X-Suno-Signature header
```

## Health Check Endpoints

```
typescript
/**
 * @route GET /api/health
 * @description Overall system health
 * @access Public
 */
interface HealthResponse {
  status: 'healthy' | 'degraded' | 'unhealthy';
  timestamp: string;
  services: {
    database: { status: 'up' | 'down'; latency: number };
    search: { status: 'up' | 'down'; latency: number };
    storage: { status: 'up' | 'down'; latency: number };
    payment: { status: 'up' | 'down'; latency: number };
  };
  version: string;
  uptime: number;
}

/**
 * @route GET /api/health/ready
 * @description Readiness probe for Kubernetes
 * @access Internal
 */
```

## Rate Limiting Configuration

```
javascript
// lib/middleware/rate-limit.ts
const rateLimitConfig = {
  'ip:anonymous': {
    windowMs: 60 * 1000, // 1 minute
    max: 20,
    message: 'Too many requests from this IP'
```

```
  },
  'user:creator': {
    windowMs: 60 * 1000,
    max: 60,
    keyGenerator: (req) => `user:${req.user.id}`
  },
  'user:venue': {
    windowMs: 60 * 1000,
    max: 1000,
    keyGenerator: (req) => `venue:${req.user.venueId}`
  },
  'endpoint:search': {
    windowMs: 60 * 1000,
    max: 100,
    keyGenerator: (req) => `search:${req.ip}`
  }
};




### **YENİ BÖLÜM: 1.5 DEPLOYMENT CHECKLIST**

## 1.5. Deployment Checklist

### Pre-Deployment (Local Development)
- [ ] All environment variables set in .env.local
- [ ] Docker containers running: PostgreSQL, Meilisearch,
Redis
- [ ] Database migrations applied: `npm run db:migrate`
- [ ] Test data seeded: `npm run db:seed`
- [ ] Search index initialized: `npm run search:init`
- [ ] All unit tests passing: `npm test`
- [ ] TypeScript compilation: `npm run type-check`
- [ ] ESLint passes: `npm run lint`

### Staging Deployment (Vercel + AWS)
- [ ] Vercel project created for each app
- [ ] Environment variables set in Vercel dashboard
- [ ] Custom domains configured (staging.aurastream.com)
- [ ] AWS resources provisioned (S3, CloudFront, Lambda)
- [ ] IAM roles configured with least privilege
- [ ] Database backup enabled (Supabase PITR)
```

- [ ] Monitoring configured (Sentry, Vercel Analytics)
- [ ] Smoke tests passing on staging

### Production Deployment
#### Phase 1: Infrastructure
- [ ] AWS production accounts secured with MFA
- [ ] S3 buckets created with versioning enabled
- [ ] CloudFront distribution with SSL certificate
- [ ] Lambda functions deployed with provisioned concurrency
- [ ] RDS/PostgreSQL with read replicas
- [ ] Redis cluster for rate limiting
- [ ] WAF configured for DDoS protection

#### Phase 2: Application
- [ ] Vercel production projects linked to Git main branch
- [ ] Production environment variables verified
- [ ] Database migrated from staging to production
- [ ] Search index populated
- [ ] CDN cache warmed up
- [ ] SSL certificates valid and renewed

#### Phase 3: Verification
- [ ] All API endpoints responding (< 500ms)
- [ ] Authentication flow working
- [ ] Payment processing test successful
- [ ] Audio streaming functional
- [ ] Search returning results
- [ ] Admin panel accessible
- [ ] Monitoring dashboards active

### Rollback Plan
```markdown
## Rollback Trigger Conditions:
1. Error rate > 5% for 5 minutes
2. Critical feature broken (payment, streaming)
3. Database migration failure
4. Security vulnerability detected

## Rollback Steps:
1. Revert Git to previous stable tag
2. Rollback database migration (if applicable)
3. Revert Vercel deployment to previous version
```

```
4. Clear CDN cache for affected paths
5. Notify users via status page
6. Investigate root cause

## Time Targets:
- Detection: < 5 minutes
- Decision: < 2 minutes
- Rollback execution: < 10 minutes
- Full recovery: < 30 minutes
```

# Post-Deployment Tasks

- [ ] Update DNS records (if applicable)
- [ ] Configure backup schedules
- [ ] Set up alert thresholds in Sentry
- [ ] Document deployment in changelog
- [ ] Notify stakeholders
- [ ] Monitor error rates for 24 hours
- [ ] Performance benchmarking

```
## 📝 **DOKÜMAN 1 İÇİN YAPILACAKLAR ÖZETİ**

### **EKLENECEK YENİ BÖLÜMLER:**
1. **1.2 Environment Variables Master List** ✓ (yukarıda)
2. **1.3 Build Order & Dependency Matrix** ✓ (yukarıda)
3. **1.4 API Endpoint Specifications** ✓ (yukarıda)
4. **1.5 Deployment Checklist** ✓ (yukarıda)

### **GÜNCELLENECEK MEVCUT BÖLÜMLER:**

#### **Bölüm 3: Search Engine – Detaylandır**
```markdown
## 3.4. Meilisearch Production Configuration

### Index Settings (production):
```json
{
  "uid": "tracks_prod",
  "primaryKey": "id",
```

```
  "searchableAttributes": [
    "title", "genre", "mood_tags", "instruments",
"description"
  ],
  "filterableAttributes": [
    "bpm", "duration", "key", "tuning", "is_instrumental",
    "genre", "mood", "language", "created_at",
"popularity_score"
  ],
  "sortableAttributes": [
    "created_at", "popularity_score", "bpm", "duration"
  ],
  "typoTolerance": {
    "enabled": true,
    "minWordSizeForTypos": { "oneTypo": 5, "twoTypos": 9 }
  },
  "faceting": { "maxValuesPerFacet": 100 },
  "pagination": { "maxTotalHits": 10000 }
}
```

## Sync Monitoring:

```
typescript
// Health check endpoints
GET /api/search/health    // Meilisearch status
GET /api/search/stats     // Index statistics
GET /api/search/sync-status  // Last sync timestamp

// Manual sync triggers
POST /api/admin/search/sync-all     // Full reindex
POST /api/admin/search/sync-track/:id  // Single track
POST /api/admin/search/repair       // Fix inconsistencies



text
#### **Bölüm 5: Operational Excellence - Detaylandır**
```markdown
## 5.7. Performance SLAs (Service Level Agreements)

### API Response Times (P95):
```
```

```
- Search queries: < 200ms
- Stream URL generation: < 100ms
- License generation: < 2s (including PDF)
- User authentication: < 500ms
- File uploads: < 5s (per MB)

### Availability Targets:
- Core services (API, DB): 99.9%
- Audio streaming: 99.95%
- Payment processing: 99.99%
- Search functionality: 99.9%

### Error Budgets (Monthly):
- Total errors: < 0.1% of requests
- 5xx errors: < 0.05% of requests
- Timeout errors: < 0.01% of requests
```

# 🛠️ DOKÜMAN 1 DÜZELTME İÇİN AI PROMPT ÖRNEKLERİ

## Environment Variables için:

```
text
"Sen bir senior DevOps mühendisisin. Şu teknolojileri
kullanan bir proje için environment variables listesi
oluştur:
- Frontend: Next.js 14 (App Router), Tailwind CSS
- Backend: Supabase (PostgreSQL, Auth, Storage), AWS (S3,
CloudFront, Lambda, SQS)
- Search: Meilisearch (Docker + production)
- Payment: Stripe (subscriptions + one-time)
- AI: Suno API (audio generation)
- Monitoring: Sentry, Vercel Analytics

Her variable için şu bilgileri içer:
1. Variable adı (NEXT_PUBLIC_... vs normal)
2. Açıklama (Türkçe)
3. Örnek değer
```

```
4. Hangi ortamda gerekli (development/production/both)
5. Güvenlik seviyesi (public/secret/sensitive)
6. Varsayılan değer (optional)

Ayrıca validation script'i ve production vs development
farklarını göster."
```

## API Spec için:

```
text
"Sen bir backend architect'sin. Müzik streaming platformu
için REST API spesifikasyonu yaz:

1. GET /api/tracks/search - Arama endpoint'i
   - Query params: q, filters, page, limit, sortBy
   - Response: paginated tracks + facets
   - Rate limiting: IP bazlı

2. GET /api/stream/:id - Audio stream URL generator
   - Authentication required
   - JWT validation
   - Returns signed CloudFront URL

3. POST /api/licenses/generate - Lisans oluşturma
   - Request body validation
   - Stripe payment integration
   - PDF generation

4. POST /api/webhooks/suno - AI webhook handler
   - Signature validation
   - Async processing

Her endpoint için: route, method, request/response schemas,
error codes, rate limiting, authentication requirements
belirt. OpenAPI 3.0 formatında."
```

## Build Order için:

```
text
```

"Sen bir monorepo expert'isin. TurboRepo kullanan bir proje için build dependency graph oluştur:

Paketler:
- packages/ui-kit (shared components)
- packages/database-client (Supabase wrapper)
- packages/search-client (Meilisearch sync)
- lib/audio/engine (Web Audio API)
- lib/services/s3 (AWS client)
- apps/creator-store (Next.js)
- apps/venue-player (Next.js PWA)
- apps/admin-factory (Next.js)

Her paket için:
1. Build sırası (1-10)
2. Bağımlılıklar (hangi paketler önce build edilmeli)
3. Output (ne üretiyor)
4. Test strategy

Ayrıca TurboRepo pipeline configuration ve build scripts ekle."

**2. DATABASE SCHEMA (NORMALIZED)**

**JSONB kullanımı kaldırılarak normalize edilmiş, analitik destekli yapı.***

**2.1. Core Tables**

`profiles`, `venues`, `tracks`, `track_files`, `track_reviews`, `licenses`, `playback_sessions`.

**profiles:** `id`, `role` ('venue', 'creator', 'admin'), `subscription_tier`.**

**venues:** `id`, `owner_id`, `business_name`, `address` (Structured), `verification_status`.**

**2.2. Track Ecosystem (Normalize Edilmiş) Search Enhancements**

**tracks:****

* `id` (UUID, PK)

**  * `title`, `bpm`, `duration_sec`, `is_instrumental`**

**  * `status`: 'pending_qc', 'processing', 'active', 'rejected'**

**  * `ai_metadata`: JSONB (Prompt, Model, Seed - Hukuki zorunluluk)**

*** **track_files:****

**  * `id` (PK), `track_id` (FK)**

**  * `file_type`: 'raw', 'stream_aac', 'stream_flac'**

**  * `tuning`: '440hz', '432hz'**

**  * `s3_key`, `lufs_value`**

*** **track_reviews:****

**  * `track_id`, `reviewer_id`, `decision`, `notes`, `reviewed_at`**

* **custom_requests:**

- `id` (UUID, PK)
- `user_id` (FK -> profiles)
- `status`: 'pending', 'processing', 'review', 'completed', 'rejected'
- `specs` (JSONB): `{ "genre": "Lo-Fi", "bpm": 90, "ref_url": "...", "notes": "..." }`
- `delivery_track_id` (FK -> tracks, nullable)
- `price_paid`: Decimal (Sipariş maliyeti)

**`saved_searches`**:

- `id` (PK), `user_id` (FK), `name` (Örn: "Sabah Kahvesi Listem").

- `query_params`: JSONB (Örn: `{ "bpm_min": 90, "mood": ["chill"], "tuning": "432hz" }`).
- `notification_enabled`: Boolean (Bu kriterlere uyan yeni şarkı gelince bildir).

**`search_logs`** (Content Gap Analizi İçin):

- `id` (PK), `query_text`, `filters_used` (JSONB).
- `result_count`: Integer (0 ise "Content Gap" alarmı üretir).
- `user_id`: UUID (Opsiyonel).
- `created_at`: Timestamp.

### 2.3. Legal & Analytics

*** **licenses:****

*** `id`, `license_key` (QR Hash), `user_id`, `project_name`**

*** `watermark_hash`: PDF içine gömülen steganografik imza.**

*** **playback_sessions:****

*** `venue_id`, `track_id`, `played_at`, `duration_listened`, `skipped`.**

### 2.4. Commercial Licensing & Analytics

```
-- İtiraz ve Destek Talepleri
create table dispute_logs (
  id uuid primary key,
  user_id uuid references profiles(id),
  license_id uuid references licenses(id),
  platform text, -- 'youtube', 'instagram'
  status text, -- 'open', 'resolved'
  created_at timestamptz
);
```

### 2.5. Search & User Preferences (YENİ TABLOLAR)

SQL

```
-- Kaydedilen Aramalar
create table saved_searches (
  id uuid primary key default gen_random_uuid(),
  user_id uuid references profiles(id),
  name text not null, -- Örn: "Hızlı Vlog Müzikleri"
  query_params jsonb not null, -- { "bpm_min": 100, "tags":
["happy"], "fuzzy": "pno" }
  notify_on_match boolean default false,
  created_at timestamptz default now()
);

-- Search Analytics (Content Gap Analizi İçin)
-- Sonuç dönmeyen veya tıklama almayan aramaları loglarız.
create table search_logs (
  id uuid primary key default gen_random_uuid(),
  user_id uuid references profiles(id), -- Anonim de
olabilir
  raw_query text,
  filters_used jsonb,
  result_count integer,
  latency_ms integer, -- Performans takibi
  created_at timestamptz default now()
);
```

# 2.6: ROW LEVEL SECURITY (RLS) POLICIES (KRİTİK)

**## 2.6. Row Level Security (RLS) Policies**

**### Enabling RLS on All Tables:**

```sql
-- Enable RLS on all sensitive tables
ALTER TABLE profiles ENABLE ROW LEVEL SECURITY;
ALTER TABLE venues ENABLE ROW LEVEL SECURITY;
ALTER TABLE tracks ENABLE ROW LEVEL SECURITY;
ALTER TABLE track_files ENABLE ROW LEVEL SECURITY;
```

```
ALTER TABLE licenses ENABLE ROW LEVEL SECURITY;
ALTER TABLE playback_sessions ENABLE ROW LEVEL SECURITY;
ALTER TABLE saved_searches ENABLE ROW LEVEL SECURITY;
```

## Profiles Table Policies:

```
sql
-- Users can only see their own profile
CREATE POLICY "Users can view own profile" ON profiles
  FOR SELECT USING (auth.uid() = id);

-- Users can update their own profile
CREATE POLICY "Users can update own profile" ON profiles
  FOR UPDATE USING (auth.uid() = id);

-- Only admins can delete profiles
CREATE POLICY "Only admins can delete profiles" ON profiles
  FOR DELETE USING (
    EXISTS (SELECT 1 FROM profiles WHERE id = auth.uid()
AND role = 'admin')
  );
```

## Tracks Table Policies:

```
sql
-- Public can view active tracks
CREATE POLICY "Active tracks are viewable by everyone" ON
tracks
  FOR SELECT USING (status = 'active');

-- Creators can view their own tracks (any status)
CREATE POLICY "Creators can view own tracks" ON tracks
  FOR SELECT USING (
    created_by = auth.uid()
    OR EXISTS (SELECT 1 FROM profiles WHERE id = auth.uid()
AND role = 'admin')
  );

-- Only admins can insert/update/delete tracks
```

```
CREATE POLICY "Only admins and creators can insert tracks"
ON tracks
  FOR INSERT WITH CHECK (
    EXISTS (
      SELECT 1 FROM profiles
      WHERE id = auth.uid()
      AND role IN ('admin', 'creator')
    )
  );

CREATE POLICY "Only admins can update/delete tracks" ON
tracks
  FOR ALL USING (
    EXISTS (SELECT 1 FROM profiles WHERE id = auth.uid()
AND role = 'admin')
  );
```

## Track Files Policies:

```
sql
-- Public can view track files for active tracks
CREATE POLICY "Public can view files for active tracks" ON
track_files
  FOR SELECT USING (
    EXISTS (
      SELECT 1 FROM tracks
      WHERE tracks.id = track_files.track_id
      AND tracks.status = 'active'
    )
  );

-- Only system (via service role) can manage track files
CREATE POLICY "Only service role can manage track files" ON
track_files
  FOR ALL USING (auth.jwt() ->> 'role' = 'service_role');
```

## Licenses Table Policies:

```
sql
```

```
-- Users can view their own licenses
CREATE POLICY "Users can view own licenses" ON licenses
  FOR SELECT USING (user_id = auth.uid());

-- Users can create licenses (through API)
CREATE POLICY "Users can create licenses" ON licenses
  FOR INSERT WITH CHECK (user_id = auth.uid());

-- System can update licenses (for watermarking, etc.)
CREATE POLICY "Service role can update licenses" ON
licenses
  FOR UPDATE USING (auth.jwt() ->> 'role' =
'service_role');
```

## Venues Table Policies:

```
sql
-- Venue owners can view their own venues
CREATE POLICY "Venue owners can view own venues" ON venues
  FOR SELECT USING (owner_id = auth.uid());

-- Venue owners can update their own venues
CREATE POLICY "Venue owners can update own venues" ON
venues
  FOR UPDATE USING (owner_id = auth.uid());

-- Admins can manage all venues
CREATE POLICY "Admins can manage all venues" ON venues
  FOR ALL USING (
    EXISTS (SELECT 1 FROM profiles WHERE id = auth.uid()
AND role = 'admin')
  );
```

## Service Role Configuration:

```
sql
-- Service role for backend operations
CREATE ROLE service_role;
GRANT ALL ON ALL TABLES IN SCHEMA public TO service_role;
```

```
-- Important: These policies DON'T apply to service_role
ALTER TABLE tracks FORCE ROW LEVEL SECURITY;
-- But exclude service_role
ALTER TABLE tracks EXCLUDE service_role FROM ROW LEVEL
SECURITY;
```

````
text
---

## ⚡ **BÖLÜM 2.7: INDEXES FOR PERFORMANCE**

```markdown
## 2.7. Performance Indexes

### Critical Search Indexes:
```sql
-- Tracks table indexes
CREATE INDEX idx_tracks_status_created ON tracks(status,
created_at DESC);
CREATE INDEX idx_tracks_bpm ON tracks(bpm) WHERE status =
'active';
CREATE INDEX idx_tracks_genre ON tracks(genre) WHERE status
= 'active';
CREATE INDEX idx_tracks_created_by ON tracks(created_by);
CREATE INDEX idx_tracks_is_instrumental ON
tracks(is_instrumental) WHERE status = 'active';

-- Partial indexes for common queries
CREATE INDEX idx_tracks_active_popular
ON tracks(popularity_score DESC)
WHERE status = 'active';

CREATE INDEX idx_tracks_recently_added
ON tracks(created_at DESC)
WHERE status = 'active';
````

# Foreign Key Indexes:

```sql
-- track_files
CREATE INDEX idx_track_files_track_id ON
track_files(track_id);
CREATE INDEX idx_track_files_file_type ON
track_files(file_type);
CREATE INDEX idx_track_files_tuning ON track_files(tuning);

-- track_reviews
CREATE INDEX idx_track_reviews_track_id ON
track_reviews(track_id);
CREATE INDEX idx_track_reviews_reviewer_id ON
track_reviews(reviewer_id);
CREATE INDEX idx_track_reviews_decision ON
track_reviews(decision);

-- licenses
CREATE INDEX idx_licenses_user_id ON licenses(user_id);
CREATE INDEX idx_licenses_track_id ON licenses(track_id);
CREATE INDEX idx_licenses_created_at ON licenses(created_at
DESC);

-- playback_sessions
CREATE INDEX idx_playback_sessions_venue_id ON
playback_sessions(venue_id);
CREATE INDEX idx_playback_sessions_track_id ON
playback_sessions(track_id);
CREATE INDEX idx_playback_sessions_played_at ON
playback_sessions(played_at DESC);
CREATE INDEX idx_playback_sessions_venue_date
ON playback_sessions(venue_id, date_trunc('day',
played_at));

-- saved_searches
CREATE INDEX idx_saved_searches_user_id ON
saved_searches(user_id);
CREATE INDEX idx_saved_searches_notify ON
saved_searches(user_id)
WHERE notify_on_match = true;

-- search_logs
CREATE INDEX idx_search_logs_created_at ON
search_logs(created_at DESC);
```

```
CREATE INDEX idx_search_logs_result_count ON
search_logs(result_count)
WHERE result_count = 0;
CREATE INDEX idx_search_logs_user_id ON
search_logs(user_id);
```

## Composite Indexes for Common Query Patterns:

```
sql
-- For "get active tracks by genre and bpm range"
CREATE INDEX idx_tracks_genre_bpm_status
ON tracks(genre, bpm, status)
WHERE status = 'active';

-- For "get user's purchased licenses"
CREATE INDEX idx_licenses_user_track
ON licenses(user_id, track_id, created_at DESC);

-- For "analytics: get venue playback by date range"
CREATE INDEX idx_playback_venue_date_range
ON playback_sessions(venue_id, played_at DESC);
```

## Index Maintenance:

```
sql
-- Weekly index maintenance job
CREATE OR REPLACE FUNCTION maintain_indexes()
RETURNS void AS $$
BEGIN
  -- Reindex heavily written tables
  REINDEX TABLE CONCURRENTLY tracks;
  REINDEX TABLE CONCURRENTLY playback_sessions;

  -- Update statistics
  ANALYZE tracks;
  ANALYZE playback_sessions;
  ANALYZE licenses;
END;
$$ LANGUAGE plpgsql;
```

```
-- Schedule with pg_cron
SELECT cron.schedule('weekly-index-maintenance', '0 2 * *
0',
  'SELECT maintain_indexes()');
```

## Index Size Monitoring:

```
sql
-- Query to monitor index sizes
SELECT
  tablename,
  indexname,
  pg_size_pretty(pg_relation_size(indexname::regclass)) as
index_size,
  idx_scan as index_scans
FROM pg_stat_user_indexes
WHERE schemaname = 'public'
ORDER BY pg_relation_size(indexname::regclass) DESC;
```

```
## 🧱 **BÖLÜM 2.8: DATABASE FUNCTIONS & COMPUTED COLUMNS**

```markdown
## 2.8. Database Functions & Business Logic

### Popularity Score Calculation:
```sql
CREATE OR REPLACE FUNCTION
calculate_popularity_score(track_id UUID)
RETURNS FLOAT AS $$
DECLARE
  play_count INTEGER;
  like_count INTEGER;
  license_count INTEGER;
  skip_rate FLOAT;
  score FLOAT;
BEGIN
  -- Get play statistics
```

```
  SELECT COUNT(*) INTO play_count
  FROM playback_sessions
  WHERE track_id = $1
    AND played_at > NOW() - INTERVAL '30 days';

  -- Get likes
  SELECT COUNT(*) INTO like_count
  FROM track_likes
  WHERE track_id = $1;

  -- Get license count
  SELECT COUNT(*) INTO license_count
  FROM licenses
  WHERE track_id = $1;

  -- Calculate skip rate (if any plays)
  IF play_count > 0 THEN
    SELECT
      CAST(COUNT(CASE WHEN skipped = true THEN 1 END) AS
FLOAT) / play_count
    INTO skip_rate
    FROM playback_sessions
    WHERE track_id = $1;
  ELSE
    skip_rate := 0;
  END IF;

  -- Calculate weighted score
  -- Weights: plays (30%), likes (40%), licenses (25%),
skip rate penalty (-5%)
  score :=
    (LEAST(play_count, 1000) / 1000.0 * 0.3) +  --
Normalize plays to 0-1
    (LEAST(like_count, 500) / 500.0 * 0.4) +    --
Normalize likes to 0-1
    (LEAST(license_count, 100) / 100.0 * 0.25) - --
Normalize licenses to 0-1
    (skip_rate * 0.05);                          -- Penalty
for skips

  -- Ensure score between 0 and 1
  score := GREATEST(0, LEAST(1, score));
```

```
  RETURN score;
END;
$$ LANGUAGE plpgsql SECURITY DEFINER;

-- Update popularity score trigger
CREATE OR REPLACE FUNCTION update_popularity_score()
RETURNS TRIGGER AS $$
BEGIN
  UPDATE tracks
  SET popularity_score =
calculate_popularity_score(NEW.track_id)
  WHERE id = NEW.track_id;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Create triggers for tables that affect popularity
CREATE TRIGGER playback_update_popularity
AFTER INSERT OR UPDATE ON playback_sessions
FOR EACH ROW EXECUTE FUNCTION update_popularity_score();

CREATE TRIGGER license_update_popularity
AFTER INSERT ON licenses
FOR EACH ROW EXECUTE FUNCTION update_popularity_score();
```

## Content Gap Detection Function:

```
sql
CREATE OR REPLACE FUNCTION detect_content_gaps()
RETURNS TABLE (
  missing_filter TEXT,
  filter_value TEXT,
  search_count INTEGER,
  last_searched TIMESTAMPTZ
) AS $$
BEGIN
  RETURN QUERY
  WITH failed_searches AS (
    SELECT
      filters_used,
      COUNT(*) as search_count,
```

```
      MAX(created_at) as last_searched
    FROM search_logs
    WHERE result_count = 0
      AND created_at > NOW() - INTERVAL '7 days'
    GROUP BY filters_used
    HAVING COUNT(*) > 5  -- Only show significant gaps
  )
  SELECT
    'bpm_range' as missing_filter,
    (filters_used->>'bpm_min') || '-' || (filters_used-
>>'bpm_max') as filter_value,
    search_count,
    last_searched
  FROM failed_searches
  WHERE filters_used ? 'bpm_min'

  UNION ALL

  SELECT
    'genre',
    filters_used->>'genre',
    search_count,
    last_searched
  FROM failed_searches
  WHERE filters_used ? 'genre'

  UNION ALL

  SELECT
    'mood',
    filters_used->>'mood',
    search_count,
    last_searched
  FROM failed_searches
  WHERE filters_used ? 'mood';
END;
$$ LANGUAGE plpgsql;
```

## License Validation Function:

sql

```
CREATE OR REPLACE FUNCTION validate_license(
  p_license_key TEXT,
  p_track_id UUID,
  p_user_id UUID DEFAULT NULL
)
RETURNS JSONB AS $$
DECLARE
  v_license licenses%ROWTYPE;
  v_is_valid BOOLEAN;
  v_message TEXT;
  v_result JSONB;
BEGIN
  -- Find the license
  SELECT * INTO v_license
  FROM licenses
  WHERE license_key = p_license_key
    AND track_id = p_track_id
    AND expires_at > NOW();

  -- Check if license exists and is valid
  IF v_license.id IS NULL THEN
    v_is_valid := FALSE;
    v_message := 'License not found or expired';
  ELSIF p_user_id IS NOT NULL AND v_license.user_id !=
p_user_id THEN
    v_is_valid := FALSE;
    v_message := 'License does not belong to this user';
  ELSE
    v_is_valid := TRUE;
    v_message := 'License is valid';

    -- Log the validation
    INSERT INTO license_validations (
      license_id, validated_at, validated_by, is_valid
    ) VALUES (
      v_license.id, NOW(), p_user_id, TRUE
    );
  END IF;

  -- Build result
  v_result := jsonb_build_object(
    'is_valid', v_is_valid,
    'message', v_message,
```

```
    'license_id', v_license.id,
    'expires_at', v_license.expires_at,
    'project_name', v_license.project_name
  );

  RETURN v_result;
END;
$$ LANGUAGE plpgsql SECURITY DEFINER;


text
---

## 🔄 **BÖLÜM 2.9: MIGRATION STRATEGY & VERSIONING**

```markdown
## 2.9. Migration Strategy & Version Control

### Migration File Structure:


/migrations/
├── 001_initial_schema/
│   ├── up.sql
│   └── down.sql
├── 002_add_ai_metadata/
│   ├── up.sql
│   └── down.sql
├── 003_add_popularity_score/
│   ├── up.sql
│   └── down.sql
└── migration_config.yaml

text
### Example Migration: Add AI Metadata
```sql
```

```
-- migrations/002_add_ai_metadata/up.sql
BEGIN;

-- Add ai_metadata column
ALTER TABLE tracks
ADD COLUMN ai_metadata JSONB NOT NULL
DEFAULT '{"model": "suno-v1", "prompt": "legacy"}';

-- Add constraint to ensure required fields
ALTER TABLE tracks
ADD CONSTRAINT ai_metadata_required_fields
CHECK (
  ai_metadata ? 'model'
  AND ai_metadata ? 'prompt'
  AND ai_metadata ? 'seed'
);

-- Update existing records
UPDATE tracks
SET ai_metadata = jsonb_build_object(
  'model', 'suno-v1',
  'prompt', COALESCE(description, 'AI generated track'),
  'seed', FLOOR(RANDOM() * 1000000)::INTEGER,
  'generated_at', created_at
)
WHERE ai_metadata = '{"model": "suno-v1", "prompt":
"legacy"}';

COMMIT;

-- migrations/002_add_ai_metadata/down.sql
BEGIN;

-- Remove constraint
ALTER TABLE tracks DROP CONSTRAINT
ai_metadata_required_fields;

-- Remove column
ALTER TABLE tracks DROP COLUMN ai_metadata;

COMMIT;
```

## Migration Configuration (migration_config.yaml):

```
yaml
version: 1
database:
  name: aurastream
  host: ${DB_HOST}
  port: 5432
  username: ${DB_USER}
  password: ${DB_PASSWORD}

migrations:
  table_name: schema_migrations
  lock_timeout: "10s"
  validate_checksums: true

rollback:
  enabled: true
  keep_last_n: 5
  auto_rollback_on_failure: true

environment_specific:
  development:
    skip_large_migrations: false
    allow_destructive_changes: true

  production:
    skip_large_migrations: true
    allow_destructive_changes: false
    require_backup_before: true
```

## Migration Runner Script (migrate.js):

```
javascript
#!/usr/bin/env node
// scripts/migrate.js

const { execSync } = require('child_process');
const fs = require('fs');
```

```
const path = require('path');

class MigrationRunner {
  async run(migrationDir, direction = 'up') {
    const migrations = fs.readdirSync(migrationDir)
      .filter(dir => fs.statSync(path.join(migrationDir,
dir)).isDirectory())
      .sort();

    for (const migration of migrations) {
      const migrationPath = path.join(migrationDir,
migration);
      const sqlFile = path.join(migrationPath, `$
{direction}.sql`);

      if (fs.existsSync(sqlFile)) {
        console.log(`Running ${migration} ($
{direction})...`);

        try {
          const sql = fs.readFileSync(sqlFile, 'utf8');
          // Execute SQL via psql or database client
          execSync(`psql -f "${sqlFile}"`, { stdio:
'inherit' });

          // Record migration
          await this.recordMigration(migration, direction);

          console.log(`✅ ${migration} completed`);
        } catch (error) {
          console.error(`❌ ${migration} failed:`,
error.message);
          throw error;
        }
      }
    }
  }

  async recordMigration(name, direction) {
    // Implementation for tracking applied migrations
  }
}
```

## Rollback Strategy:

```
sql
-- Emergency rollback procedure
-- 1. Stop application traffic
-- 2. Run latest down.sql files in reverse order
-- 3. Verify data integrity
-- 4. Restore from backup if needed

-- Automated rollback detection
CREATE OR REPLACE FUNCTION detect_migration_issues()
RETURNS TRIGGER AS $$
BEGIN
  -- If error rate spikes after migration
  IF (SELECT COUNT(*) FROM api_errors
      WHERE occurred_at > NOW() - INTERVAL '5 minutes') >
100 THEN
    -- Alert and suggest rollback
    RAISE LOG 'High error rate detected after migration.
Consider rollback.';
  END IF;
  RETURN NULL;
END;
$$ LANGUAGE plpgsql;
```

## Database Version Tracking:

```
sql
-- Table to track schema versions
CREATE TABLE schema_versions (
  version VARCHAR(50) PRIMARY KEY,
  description TEXT,
  applied_at TIMESTAMPTZ DEFAULT NOW(),
  checksum VARCHAR(64),
  applied_by TEXT DEFAULT CURRENT_USER
);

-- Function to check current version
```

```
CREATE OR REPLACE FUNCTION get_schema_version()
RETURNS TABLE(version TEXT, applied_at TIMESTAMPTZ) AS $$
BEGIN
  RETURN QUERY
  SELECT v.version, v.applied_at
  FROM schema_versions v
  ORDER BY v.applied_at DESC
  LIMIT 1;
END;
$$ LANGUAGE plpgsql;
```

# 2.7: PERFORMANCE INDEXES

markdown
**## 2.7. Performance Indexes**

**### Index Strategy Overview:**
Indexes are critical for query performance in our heavily read-oriented application.
We prioritize:
1. **Search performance** - Fast filtering and sorting
2. **Analytics queries** - Efficient aggregation
3. **User-specific data** - Quick retrieval of user-owned content
4. **Real-time updates** - Fast joins and lookups

**### Critical Search Indexes for Tracks Table:**
```sql
-- Primary index for active tracks (most common query)
CREATE INDEX idx_tracks_status_active ON tracks(status)
WHERE status = 'active';

-- BPM range queries (common filter)
CREATE INDEX idx_tracks_bpm_range ON tracks(bpm)
WHERE status = 'active' AND bpm BETWEEN 60 AND 180;

-- Genre-based filtering
CREATE INDEX idx_tracks_genre_status ON tracks(genre, status)
WHERE status = 'active';
```

```
-- Composite index for common search patterns: genre + bpm
+ status
CREATE INDEX idx_tracks_genre_bpm_status ON tracks(genre,
bpm, status);

-- For "newest tracks" queries
CREATE INDEX idx_tracks_created_desc ON tracks(created_at
DESC)
WHERE status = 'active';

-- For popularity-based sorting
CREATE INDEX idx_tracks_popularity_desc ON
tracks(popularity_score DESC NULLS LAST)
WHERE status = 'active';

-- Instrumental filter (common use case)
CREATE INDEX idx_tracks_instrumental ON
tracks(is_instrumental, status)
WHERE status = 'active';

-- Frequency/tuning filter
CREATE INDEX idx_tracks_base_frequency ON
tracks(base_frequency)
WHERE status = 'active';

-- Partial index for pending QC (admin queries)
CREATE INDEX idx_tracks_pending_qc ON tracks(status)
WHERE status = 'pending_qc';
```

## Foreign Key and Relationship Indexes:

```
sql
-- TRACK_FILES table (high read volume)
CREATE INDEX idx_track_files_track_id ON
track_files(track_id);
CREATE INDEX idx_track_files_file_type ON
track_files(file_type);
CREATE INDEX idx_track_files_tuning ON track_files(tuning);
CREATE INDEX idx_track_files_s3_key ON track_files(s3_key);

-- Composite index for common file retrieval
```

```
CREATE INDEX idx_track_files_track_type_tuning ON
track_files(track_id, file_type, tuning);

-- TRACK_REVIEWS table
CREATE INDEX idx_track_reviews_track_id ON
track_reviews(track_id);
CREATE INDEX idx_track_reviews_reviewer_id ON
track_reviews(reviewer_id);
CREATE INDEX idx_track_reviews_decision ON
track_reviews(decision);
CREATE INDEX idx_track_reviews_reviewed_at ON
track_reviews(reviewed_at DESC);

-- For finding pending reviews
CREATE INDEX idx_track_reviews_pending ON
track_reviews(track_id)
WHERE decision IS NULL;

-- LICENSES table (heavily queried for validation)
CREATE INDEX idx_licenses_user_id ON licenses(user_id);
CREATE INDEX idx_licenses_track_id ON licenses(track_id);
CREATE INDEX idx_licenses_license_key ON
licenses(license_key);
CREATE INDEX idx_licenses_created_at ON licenses(created_at
DESC);
CREATE INDEX idx_licenses_expires_at ON
licenses(expires_at);

-- Composite index for license validation queries
CREATE INDEX idx_licenses_key_track_user ON
licenses(license_key, track_id, user_id);

-- For finding expired licenses
CREATE INDEX idx_licenses_expired ON licenses(expires_at)
WHERE expires_at < NOW();

-- PLAYBACK_SESSIONS table (analytics - high write volume)
CREATE INDEX idx_playback_sessions_venue_id ON
playback_sessions(venue_id);
CREATE INDEX idx_playback_sessions_track_id ON
playback_sessions(track_id);
CREATE INDEX idx_playback_sessions_played_at ON
playback_sessions(played_at DESC);
```

```
-- Composite indexes for analytics queries
CREATE INDEX idx_playback_venue_date ON
playback_sessions(venue_id, date_trunc('day', played_at));
CREATE INDEX idx_playback_track_date ON
playback_sessions(track_id, date_trunc('day', played_at));

-- For skip rate analysis
CREATE INDEX idx_playback_skipped ON
playback_sessions(skipped, played_at);

-- For session duration analysis
CREATE INDEX idx_playback_duration ON
playback_sessions(duration_listened)
WHERE duration_listened > 30;
```

## User Data and Search Preference Indexes:

```
sql
-- SAVED_SEARCHES table
CREATE INDEX idx_saved_searches_user_id ON
saved_searches(user_id);
CREATE INDEX idx_saved_searches_name ON
saved_searches(name);
CREATE INDEX idx_saved_searches_notify_active ON
saved_searches(user_id, notify_on_match)
WHERE notify_on_match = true;
CREATE INDEX idx_saved_searches_created ON
saved_searches(created_at DESC);

-- For finding users to notify about new matches
CREATE INDEX idx_saved_searches_notify_trigger ON
saved_searches(
  user_id,
  (query_params->>'bpm_min')::INT,
  (query_params->>'bpm_max')::INT,
  notify_on_match
) WHERE notify_on_match = true;

-- SEARCH_LOGS table (analytics - high write volume)
```

```
CREATE INDEX idx_search_logs_created_at ON
search_logs(created_at DESC);
CREATE INDEX idx_search_logs_user_id ON
search_logs(user_id)
WHERE user_id IS NOT NULL;
CREATE INDEX idx_search_logs_result_count ON
search_logs(result_count);

-- For content gap analysis (result_count = 0)
CREATE INDEX idx_search_logs_no_results ON
search_logs(created_at)
WHERE result_count = 0;

-- For analyzing popular search terms
CREATE INDEX idx_search_logs_query_text ON search_logs
USING gin(to_tsvector('english', query_text));

-- For filter usage analytics
CREATE INDEX idx_search_logs_filters ON search_logs USING
gin(filters_used);
```

## Profiles and Venues Indexes:

```
sql
-- PROFILES table
CREATE INDEX idx_profiles_email ON profiles(email);
CREATE INDEX idx_profiles_role ON profiles(role);
CREATE INDEX idx_profiles_subscription_tier ON
profiles(subscription_tier);
CREATE INDEX idx_profiles_stripe_customer ON
profiles(stripe_customer_id)
WHERE stripe_customer_id IS NOT NULL;

-- Composite index for user type queries
CREATE INDEX idx_profiles_role_tier ON profiles(role,
subscription_tier);

-- VENUES table
CREATE INDEX idx_venues_owner_id ON venues(owner_id);
CREATE INDEX idx_venues_verification_status ON
venues(verification_status);
```

```
CREATE INDEX idx_venues_business_name ON
venues(business_name);
CREATE INDEX idx_venues_created_at ON venues(created_at
DESC);

-- For finding unverified venues
CREATE INDEX idx_venues_pending_verification ON
venues(verification_status)
WHERE verification_status = 'pending';
```

## Specialized Indexes for Complex Queries:

```
sql
-- For "similar tracks" queries (genre + bpm + mood)
CREATE INDEX idx_tracks_similarity ON tracks(
  genre,
  bpm,
  (mood_tags[1]) -- First mood tag
) WHERE status = 'active';

-- For seasonal/temporal queries
CREATE INDEX idx_tracks_seasonal ON tracks(
  EXTRACT(MONTH FROM created_at),
  genre
) WHERE status = 'active';

-- For admin analytics dashboard
CREATE INDEX idx_tracks_admin_metrics ON tracks(
  status,
  created_by,
  created_at
) INCLUDE (bpm, genre); -- Covering index for common admin
queries

-- For bulk operations during sync
CREATE INDEX idx_tracks_sync_status ON tracks(status,
updated_at);
```

## Index Maintenance and Monitoring:

```sql
-- Function to identify missing indexes
CREATE OR REPLACE FUNCTION find_missing_indexes()
RETURNS TABLE(
  table_name TEXT,
  query TEXT,
  avg_cost FLOAT,
  recommendation TEXT
) AS $$
BEGIN
  RETURN QUERY
  SELECT
    schemaname || '.' || relname as table_name,
    query,
    avg_cost,
    'Consider adding index on: ' || indexable_columns as
recommendation
  FROM pg_stat_statements
  JOIN pg_class ON pg_class.oid = (regexp_matches(query,
'FROM\s+([\w\.]+)'))[1]::regclass
  WHERE calls > 100 AND avg_cost > 1000
  ORDER BY total_time DESC
  LIMIT 10;
END;
$$ LANGUAGE plpgsql;

-- Weekly index maintenance job
CREATE OR REPLACE PROCEDURE perform_index_maintenance()
LANGUAGE plpgsql
AS $$
BEGIN
  -- Reindex only heavily fragmented indexes
  REINDEX TABLE CONCURRENTLY tracks;
  REINDEX TABLE CONCURRENTLY playback_sessions;
  REINDEX TABLE CONCURRENTLY search_logs;

  -- Update statistics on frequently updated tables
  ANALYZE tracks;
  ANALYZE playback_sessions;
  ANALYZE search_logs;
  ANALYZE licenses;

  -- Log maintenance activity
```

```
  INSERT INTO maintenance_logs (operation, affected_tables,
duration)
  VALUES ('index_maintenance',
'tracks,playback_sessions,search_logs,licenses', NULL);
END;
$$;

-- Schedule index maintenance (every Sunday at 2 AM)
SELECT cron.schedule(
  'weekly-index-maintenance',
  '0 2 * * 0', -- Sunday 2 AM
  'CALL perform_index_maintenance()'
);

-- Monitoring query for index usage and bloat
SELECT
  schemaname,
  tablename,
  indexname,
  pg_size_pretty(pg_relation_size(indexrelid)) as
index_size,
  idx_scan as index_scans,
  idx_tup_read as tuples_read,
  idx_tup_fetch as tuples_fetched,
  -- Calculate bloat estimate
  pg_stat_get_dead_tuples(idx.relid) as dead_tuples
FROM pg_stat_user_indexes stat
JOIN pg_index i ON stat.indexrelid = i.indexrelid
JOIN pg_class idx ON idx.oid = i.indexrelid
WHERE schemaname = 'public'
ORDER BY pg_relation_size(indexrelid) DESC
LIMIT 20;

-- Index hit rate monitoring
SELECT
  sum(idx_blks_hit) / nullif(sum(idx_blks_hit +
idx_blks_read), 0) as hit_ratio
FROM pg_statio_user_indexes
WHERE schemaname = 'public';
```

## Index Creation Guidelines:

1. For tables < 10,000 rows: Index only critical columns
2. For tables 10,000 - 1M rows: Add indexes for common query patterns
3. For tables > 1M rows: Use partial indexes and include clauses
4. Always test: New indexes on staging before production
5. Monitor impact: Use EXPLAIN ANALYZE for query plans

## Index Naming Convention:

```
text
idx_[table]_[columns]_[condition]
Examples:
- idx_tracks_status_active
- idx_playback_venue_date
- idx_licenses_expired
```

## Performance Targets:

- Query response time: < 100ms for 95th percentile
- Index hit ratio: > 99%
- Index size: < 30% of table size
- Maintenance window: < 30 minutes weekly

```
text
---

## ✅ **2.7 BÖLÜMÜ İÇİN ÖZET:**

**Eklemen gerekenler:**
1. Search Indexes (tracks tablosu için)
2. Foreign Key Indexes (ilişkisel tablolar)
3. User Data Indexes (profiles, saved_searches)
4. Analytics Indexes (playback_sessions, search_logs)
5. Specialized Indexes (complex queries için)
6. Index Maintenance Procedures
7. Monitoring Queries

**Önemli Not:** Bu indexlerin tamamını bir kerede
oluşturma. Production'a deploy ederken:
```

1. Önce en kritik indexleri oluştur (search performance için)
2. Monitor et
3. Sonra analytics indexlerini ekle

## 2.8. Database Functions & Business Logic

### Functional Requirements Specification:

#### 1. Popularity Score Calculation
**Purpose:** Calculate dynamic popularity score (0–1) for tracks based on engagement

**Input:** `track_id` (UUID)
**Output:** `popularity_score` (DECIMAL 0.0000 – 1.0000)

**Business Rules:**
- **Weighting:**
  - Plays (last 30 days): 35% weight
  - Total likes: 40% weight
  - Licenses (last 90 days): 25% weight
- **Normalization:**
  - Plays capped at 1000 for scoring
  - Likes capped at 500 for scoring
  - Licenses capped at 100 for scoring
- **Penalties:**
  - Skip rate reduces score (50% max penalty)
  - Track age factor: newer tracks get 0.7–1.0 multiplier
- **Update triggers:** On playback, like, license events

**Performance:** Must complete in < 100ms per track

---

#### 2. License Generation
**Purpose:** Generate unique license with watermark for track purchase

**Input Parameters:**
- `user_id`, `track_id`, `project_name` (min 3 chars)
- `usage_type`: ['youtube', 'podcast', 'advertisement', 'twitch', 'instagram', 'film']

- `formats`: array of ['mp3', 'flac', 'wav']
- `frequencies`: array of [440, 432]

**Output:**
- `license_id` (UUID)
- `license_key` (32-char hex)
- `download_url` (7-day expiry)
- `pdf_url` (license certificate)
- `asset_id` (format: "ALS-YYYY-MM-XXXXXXX")

**Business Rules:**
1. Validate track is active
2. Generate unique watermark hash: `SHA256(user_id + track_id + project_name + timestamp)`
3. Asset ID must be unique and contain track/user reference
4. Download token valid for 7 days
5. Store all selected formats/frequencies in license record

**Error Cases:**
- Invalid project name → Exception
- Track not active → Exception
- Duplicate license for same project → Warning

---

#### 3. License Validation
**Purpose:** Validate license for streaming/download access

**Input:** `license_key`, `track_id`, `requested_format`, `requested_frequency`
**Output:** JSON with access decision and metadata

**Validation Rules:**
1. License must exist and not be expired
2. License must match track_id
3. Requested format must be in license.formats array
4. Requested frequency must be in license.frequencies array
5. Corresponding track_file must exist in S3

**Response Structure:**
```json
{
  "valid": boolean,
```

```
  "access_granted": boolean,
  "message": string,
  "s3_key": string|null,
  "available_formats": string[],
  "available_frequencies": number[]
}
```

Audit: Log every validation attempt

4. Content Gap Analysis

Purpose: Identify missing content based on unsuccessful searches

Input: `lookback_days` (default: 7), `min_search_count` (default: 5)
Output: List of content gaps with recommendations

Analysis Logic:

1. Find searches with `result_count = 0` in period
2. Group by filter combinations with minimum count
3. Categorize gaps:
    - BPM range gaps
    - Genre gaps
    - Mood gaps
    - Instrument gaps
    - Duration gaps

Output Columns:

- `gap_type`, `filter_name`, `filter_value`
- `search_count`, `last_searched`
- `recommended_action` (human-readable suggestion)

Example Recommendation Logic:

- BPM < 60 → "Produce slower tempo tracks"

- Genre = "Cinematic" → "Focus on cinematic genre next batch"
- No 432Hz versions → "Generate 432Hz versions for popular tracks"

5. Smart Cache Management

Purpose: Manage offline cache for venue players

Input: `venue_id`, `available_space_mb` (default: 500)
Output: List of tracks to cache/evict

Caching Algorithm:

1. Priority factors:
    - Scheduled for upcoming plays (highest)
    - Recent play frequency
    - Liked by venue
    - Track popularity score
    - File size (optimize space)
2. Eviction policy (LRU + weighted):
    - Not played in last 30 days (first)
    - Low popularity score
    - Large file size relative to value

Output:

- `action`: ['cache', 'keep', 'evict']
- `track_id`, `priority_score`, `file_size_mb`, `reason`

## Implementation Notes for AI:

Security Considerations:

- All financial/license functions use `SECURITY DEFINER`
- Input validation at database level

- Audit logging for sensitive operations

Performance Guidelines:

- Use partial indexes for date-range queries
- Implement batch operations for bulk updates
- Consider materialized views for complex analytics

Error Handling:

- Use custom exception messages
- Include context in error logs
- Graceful degradation when possible

Testing Requirements:

- Unit tests for scoring algorithm edge cases
- Concurrency tests for license generation
- Performance tests for analytics functions

# 2.9: MIGRATION STRATEGY & VERSIONING

```
markdown
## 2.9. Migration Strategy & Version Control

### Overview:
Database schema evolution management system supporting
zero-downtime deployments, rollback capabilities, and
version tracking.

### 1. Migration File Structure Specification:

**Directory Layout:**
```

```
/migrations/
 ├── versions.yaml # Version manifest
 ├── 001_initial_schema/ # Each migration in separate folder
 |   ├── up.sql # Forward migration
 |   ├── down.sql # Rollback script
```

```
│   └── metadata.json # Migration metadata
├── 002_add_ai_metadata/
│   ├── up.sql
│   ├── down.sql
│   └── metadata.json
└── tools/
    ├── migrate.ts # Migration runner
    ├── rollback.ts # Rollback utility
    └── validate.ts # Schema validation
```

```
text
**File Naming Convention:**
- `{3-digit_sequence}_{descriptive_name}/`
- Sequence: Incremental, gaps allowed for branching
- Descriptive: Lowercase with underscores

---

### 2. Migration Types & Patterns:

#### Type A: Schema Changes (Safe)
- Adding nullable columns
- Adding indexes (concurrently)
- Adding CHECK constraints (VALIDATE after)
- Creating new tables

#### Type B: Data Migrations (Risky)
- Backfilling data
- Transforming existing data
- Adding NOT NULL constraints (after backfill)
- Dropping columns (soft delete first)

#### Type C: Destructive Changes (High Risk)
- Dropping tables/columns
- Changing column types
- Adding UNIQUE constraints to existing data

---
```

### 3. Zero-Downtime Deployment Rules:

**Rule 1: Backward Compatibility**
- New code must work with old schema
- Old code must work with new schema (during deployment)
- Use feature flags for breaking changes

**Rule 2: Migration Order**
1. Add new nullable columns/tables
2. Deploy new application code
3. Backfill/transform data
4. Add constraints/indexes
5. Remove deprecated columns (after verification)

**Rule 3: Rollback Preparedness**
- Every `up.sql` must have corresponding `down.sql`
- Test rollback on staging before production
- Keep last 5 migrations easily revertible

---

### 4. Migration Metadata Specification:

**metadata.json Structure:**
```json
{
  "id": "002_add_ai_metadata",
  "description": "Add AI metadata column to tracks table",
  "type": "schema_change",
  "risk_level": "low",
  "estimated_duration": "5s",
  "requires_maintenance_window": false,
  "dependencies": ["001_initial_schema"],
  "validations": [
    "check_tracks_ai_metadata_not_null",
    "verify_ai_metadata_json_structure"
  ],
  "rollback_strategy": "immediate",
  "notify_on": ["start", "success", "failure"]
}
```

## 5. Migration Execution Workflow:

Pre-Migration Checklist:

- Backup database (point-in-time)
- Verify disk space (> 2x migration size)
- Check replication lag (< 1s)
- Notify monitoring team
- Set maintenance page (if needed)

Migration Steps:

1. Pre-flight validation
    - Check dependencies
    - Validate SQL syntax
    - Estimate resource usage
2. Dry-run on staging
    - Execute with `--dry-run` flag
    - Verify no data loss
    - Check performance impact
3. Production execution
    - Start transaction
    - Apply migration
    - Run validations
    - Commit/rollback based on results
4. Post-migration verification
    - Health check APIs
    - Monitor error rates
    - Verify data integrity
    - Update schema version

## 6. Rollback Strategy:

Automatic Rollback Triggers:

- Migration runtime > 5x estimated duration
- Error rate increase > 5% post-migration
- Critical business function failure
- Data integrity violation detected

Rollback Priority Levels:

- P0 (Immediate): Data corruption risk
- P1 (Within 15 min): Service degradation
- P2 (Next maintenance): Non-critical issues

Rollback Execution Steps:

1. Stop application writes (if possible)
2. Execute `down.sql` in reverse order
3. Verify schema consistency
4. Restore from backup (if rollback fails)
5. Notify stakeholders

## 7. Version Control & Tracking:

Schema Versions Table:

```
sql
-- Auto-created by migration system
CREATE TABLE IF NOT EXISTS schema_migrations (
  version VARCHAR(50) PRIMARY KEY,
  description TEXT NOT NULL,
  checksum CHAR(64) NOT NULL,  -- SHA256 of up.sql
  executed_at TIMESTAMPTZ DEFAULT NOW(),
  execution_duration INTERVAL,
  executed_by TEXT DEFAULT CURRENT_USER,
  success BOOLEAN NOT NULL,
  rollback_attempts INTEGER DEFAULT 0
);
```

Version Manifest (versions.yaml):

```yaml
current_version: "002_add_ai_metadata"
applied_migrations:
  - id: "001_initial_schema"
    applied_at: "2024-01-15T10:30:00Z"
    checksum: "abc123..."
  - id: "002_add_ai_metadata"
    applied_at: "2024-01-20T14:45:00Z"
    checksum: "def456..."
pending_migrations: []
compatibility:
  min_app_version: "1.2.0"
  max_app_version: "2.0.0"
```

## 8. Testing & Validation Requirements:

Unit Testing:

- Each migration must have test data
- Verify `down.sql` reverts `up.sql` completely
- Test with production-like data volume

Integration Testing:

- Apply migration sequence on test environment
- Verify application functionality
- Performance benchmark comparison

Canary Testing:

- Apply to 10% of production traffic first
- Monitor metrics for 1 hour
- Rollback if anomalies detected

## 9. Monitoring & Alerting:

Key Metrics to Monitor:

- Migration duration vs estimate
- Database locks held
- Replication lag during migration
- Error rate during/post migration
- Rollback frequency

Alert Conditions:

- Migration running > 30 minutes
- Rollback triggered
- Schema inconsistency detected
- Version mismatch between nodes

## 10. Tooling Requirements:

Migration Runner Must Support:

- Transaction management (commit/rollback)
- Concurrent index creation
- Dry-run mode
- Progress reporting
- Parallel execution (safe operations)
- Dependency resolution
- Checksum validation

CLI Interface:

```
bash
# Basic commands
npm run db:migrate -- --to-version=003
npm run db:rollback -- --to-version=001
npm run db:status                  # Show migration status
npm run db:validate                # Validate pending
migrations
npm run db:create -- --name=add_feature_x  # Generate
migration template
```

## 11. Team Coordination:

Communication Protocol:

1. 24 hours before: Notify all engineers
2. 1 hour before: Final readiness check
3. During: Real-time status updates
4. After: Post-mortem for any issues

Responsibility Matrix:

- Migration Author: Write up/down scripts, tests
- Database Lead: Review performance impact
- Release Manager: Schedule and coordinate
- SRE: Monitor and respond to issues

## 12. Emergency Procedures:

Migration Failure Scenarios:

Scenario 1: Migration hangs

- Action: Kill long-running queries after timeout
- Fallback: Manual intervention by DBA
- Communication: Immediate team notification

Scenario 2: Data corruption detected

- Action: Immediate rollback
- Fallback: Restore from backup
- Investigation: Root cause analysis required

Scenario 3: Application incompatibility

- Action: Rollback migration
- Fallback: Deploy compatible app version first
- Lesson: Better compatibility testing needed

Success Criteria:

- Zero data loss
- Under 5 minutes downtime (if any)
- All applications functional post-migration
- Monitoring confirms system health
- Rollback tested and verified

```
text
---

## ✅ **BÖLÜM 2.9 ÖZETİ:**

**Kapsadığımız spesifikasyonlar:**
1. File structure & naming conventions
2. Migration types & risk levels
3. Zero-downtime deployment rules
4. Metadata requirements
5. Execution workflow
6. Rollback strategy
7. Version tracking
8. Testing requirements
9. Monitoring & alerting
10. Tooling specs
11. Team coordination
12. Emergency procedures

**AI'ın yapması gereken:**
- Migration runner tool'u implement etmek
- SQL script'leri oluşturmak
- Validation logic yazmak
- Monitoring integration kurmak
```

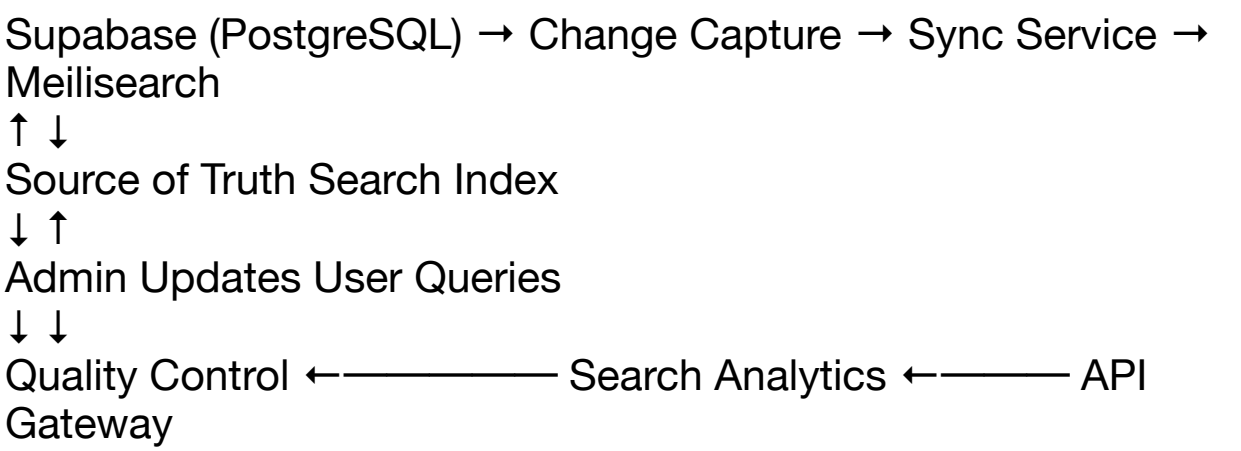
# 3: ADVANCED SEARCH ENGINE (MEILISEARCH)

```
markdown
## 3. ADVANCED SEARCH ENGINE

### 3.1. Architecture Overview
```

**System Components:**
1. **Primary Database:** Supabase PostgreSQL (source of truth)
2. **Search Engine:** Meilisearch (read-optimized index)
3. **Sync Layer:** Real-time change capture + batch processing
4. **Query Layer:** REST API with caching and rate limiting
5. **Analytics:** Search log collection and analysis

**Data Flow:**




```
text
---

### 3.2. Meilisearch Configuration Specification

**Index Configuration Requirements:**

**Index Name:** `tracks_production` (environment-based naming)

**Searchable Attributes:**
- `title` (highest priority)
- `genre`
- `mood_tags` (array)
- `instruments` (array)
- `description` (optional, lower priority)
```

**Filterable Attributes:**
- `bpm` (integer range)
- `duration_sec` (integer range)
- `key` (string: 'C', 'Dm', etc.)
- `tuning` (integer: 440, 432, 528)
- `is_instrumental` (boolean)
- `genre` (string)
- `mood` (string)
- `language` (string)
- `created_at` (timestamp)
- `popularity_score` (float)
- `base_frequency` (integer)

**Sortable Attributes:**
- `created_at` (descending by default)
- `popularity_score` (descending)
- `bpm` (ascending)
- `duration_sec` (ascending)

**Ranking Rules:**
1. `words` (exact match first)
2. `typo` (typo tolerance)
3. `proximity` (term proximity)
4. `attribute` (field importance)
5. `popularity:desc` (engagement-based)
6. `exactness` (exact matches)

**Typo Tolerance Settings:**
- Enabled: true
- Min word size for 1 typo: 5 characters
- Min word size for 2 typos: 9 characters
- Disabled for: ['bpm', 'duration', 'key'] (numeric/exact fields)

**Faceting Configuration:**
- Max values per facet: 100
- Sort facet values by: `count:desc`
- Facet attributes: ['genre', 'mood', 'instruments', 'key', 'tuning']

---

### 3.3. Data Synchronization Specification

**Sync Triggers:**
1. **Real-time:** PostgreSQL changes via Supabase webhooks
2. **Scheduled:** Hourly full consistency check
3. **Manual:** Admin-triggered reindex

**Change Detection Rules:**
- **Insert:** When track status changes to 'active'
- **Update:** When active track metadata changes
- **Delete:** When track status changes from 'active' or track is deleted
- **Ignore:** Updates to non-searchable fields (internal metadata)

**Sync Payload Transformation:**

**Source (PostgreSQL) → Target (Meilisearch Document):**
```typescript
interface SyncTransformation {
  // Direct mappings
  id: track.id,
  title: track.title,
  bpm: track.bpm,
  duration: track.duration_sec,

  // Array transformations
  mood_tags: track.mood_tags || [],
  instruments: track.instruments || [],

  // Type conversions
  is_instrumental: Boolean(track.is_instrumental),
  tuning: track.base_frequency,

  // Computed fields
  popularity_score: track.popularity_score || 0,

  // Date formatting
  created_at: track.created_at.toISOString(),

  // Categorization
  genre: track.genre,
  key: track.key,
  language: track.language || 'instrumental'
```

```
}
```

Sync Service Requirements:

Performance Requirements:

- Latency: < 500ms from DB change to searchable
- Throughput: 1000 documents/minute
- Error rate: < 0.1%

Reliability Requirements:

- At-least-once delivery guarantee
- Dead letter queue for failed syncs
- Automatic retry with exponential backoff
- Alert after 3 consecutive failures

Monitoring Metrics:

- Sync queue depth
- Processing latency (p50, p95, p99)
- Success/failure rate
- Document freshness (index lag time)

## 3.4. Search API Specification

Endpoint: `GET /api/v1/search`

Query Parameters:

- `q` (string, optional): Search query text
- `filters` (string, optional): JSON string of filter criteria
- `page` (integer, default: 1): Pagination page number
- `limit` (integer, default: 20, max: 100): Results per page
- `sortBy` (string, optional): 'relevance', 'newest', 'popular', 'bpm_asc', 'bpm_desc'

- `facets` (string, optional): Comma-separated facet names to return

Filter Syntax Examples:

```
json
// Single filter
"filters": "genre = 'Acoustic'"

// Multiple filters (AND)
"filters": "genre = 'Acoustic' AND bpm >= 90 AND bpm <=
120"

// OR conditions
"filters": "genre = 'Acoustic' OR genre = 'Electronic'"

// Array contains
"filters": "mood_tags IN ['chill', 'relaxing']"

// Range filter
"filters": "bpm 90 TO 120 AND duration_sec 120 TO 300"

// Frequency filter
"filters": "tuning = 432 OR tuning = 440"

// Instrumental only
"filters": "is_instrumental = true"
```

Response Format:

```
json
{
  "query": {
    "text": "acoustic guitar",
    "filters": "genre = 'Acoustic'",
    "page": 1,
    "limit": 20,
    "totalResults": 150
  },
  "results": [
    {
      "id": "track_uuid",
```

```
      "title": "Sunset Boulevard",
      "genre": "Acoustic",
      "bpm": 95,
      "duration": 252,
      "popularity": 0.85,
      "formats": ["aac", "flac"],
      "frequencies": [440, 432],
      "preview_url": "/preview/track_uuid/30"
    }
  ],
  "facets": {
    "genre": [
      {"value": "Acoustic", "count": 45},
      {"value": "Electronic", "count": 32}
    ],
    "bpm": {"min": 60, "max": 180},
    "duration": {"min": 60, "max": 600},
    "mood": [
      {"value": "chill", "count": 67},
      {"value": "upbeat", "count": 43}
    ]
  },
  "pagination": {
    "currentPage": 1,
    "totalPages": 8,
    "totalResults": 150,
    "nextPage": "/api/v1/search?q=acoustic&page=2"
  }
}
```


Error Responses:

- `400 Bad Request`: Invalid filter syntax
- `429 Too Many Requests`: Rate limit exceeded
- `502 Bad Gateway`: Search engine unavailable
- `504 Gateway Timeout`: Search timeout (> 2 seconds)

## 3.5. Query Processing & Optimization

Query Parsing Rules:

1. Natural language queries: "chill acoustic guitar music for cafe"
    - Extract: genre='Acoustic', mood_tags CONTAINS 'chill', instruments CONTAINS 'guitar'
    - Add: usage_context='cafe'
2. BPM-based queries: "95 bpm acoustic"
    - Parse numeric BPM values
    - Apply ±5 BPM tolerance if exact match not found
3. Duration queries: "3 minute background music"
    - Convert to seconds range (±30 seconds tolerance)

Query Expansion Rules:

- "acoustic" → ALSO search: "guitar", "piano", "folk"
- "chill" → ALSO search: "relaxing", "calm", "peaceful"
- "upbeat" → ALSO search: "energetic", "happy", "positive"

Spell Correction:

- "acostic" → "acoustic"
- "instramental" → "instrumental"
- "elektronic" → "electronic"

Fallback Strategies:

1. Zero results: Broaden filters, remove least important constraints
2. Low results (< 5): Suggest similar genres/moods
3. Timeout: Return cached results or partial results

## 3.6. Caching Strategy

Cache Layers:

1. CDN Cache: Static filter combinations (1 hour TTL)

2. Application Cache: Frequent query results (5 minute TTL)
3. Browser Cache: User-specific searches (session-based)

Cache Invalidation Rules:

- Invalidate on: New track added, track metadata updated
- Partial invalidation: By genre, mood, or other facet values
- Grace period: Serve stale cache while reindexing (max 30 seconds)

Cache Key Composition:

```
text
search:{query_hash}:{filters_hash}:{page}:{limit}:{sort}:
{user_tier}
```

Performance Targets:

- Cache hit rate: > 80% for common queries
- Cache response time: < 50ms
- Cache size limit: 10,000 entries (LRU eviction)

## 3.7. Search Analytics Specification

Data Collection Points:

1. Query logging: All search requests
2. Result interactions: Clicks, previews, skips
3. Zero-result queries: Failed searches
4. Filter usage: Which filters are applied
5. Performance metrics: Response times, cache hits

Analytics Schema:

```
sql
-- Extended search_logs table requirements
ALTER TABLE search_logs ADD COLUMN IF NOT EXISTS session_id
UUID;
```

```
ALTER TABLE search_logs ADD COLUMN IF NOT EXISTS
response_time_ms INTEGER;
ALTER TABLE search_logs ADD COLUMN IF NOT EXISTS result_ids
UUID[];
ALTER TABLE search_logs ADD COLUMN IF NOT EXISTS
clicked_result_id UUID;
ALTER TABLE search_logs ADD COLUMN IF NOT EXISTS
click_position INTEGER;
```

Content Gap Detection:

- Trigger: Search with 0 results
- Analysis: Weekly report of top missing content
- Action: Feed to AI generation pipeline

Trend Analysis:

- Daily/weekly popular search terms
- Seasonal trends (holiday music, summer vibes)
- Emerging genre/mood preferences

A/B Testing Support:

- Track different ranking algorithms
- Measure click-through rate improvements
- Test new filter combinations

## 3.8. Performance & Scaling Requirements

Response Time SLAs:

- P50: < 100ms
- P95: < 200ms
- P99: < 500ms
- Timeout: 2 seconds

Concurrent Capacity:

- Expected peak: 1000 concurrent searches

- Maximum supported: 5000 concurrent searches
- Rate limit per IP: 60 requests/minute
- Rate limit per user: 100 requests/minute

Scaling Strategy:

- Vertical scaling: Increase Meilisearch memory/CPU
- Horizontal scaling: Read replicas for search
- Sharding: By genre or creation date if > 1M documents

Resource Requirements:

- Memory: 2GB + (document_count × 10KB)
- Disk: 5GB + (document_count × 50KB)
- CPU: 2 cores minimum, 4 cores recommended

## 3.9. Monitoring & Alerting

Health Checks:

- `/health/search`: Meilisearch connectivity
- `/health/sync`: Sync service status
- `/health/cache`: Cache hit rates

Key Metrics to Monitor:

1. Availability: Uptime, error rates
2. Performance: Response times, latency
3. Accuracy: Click-through rates, zero-result rate
4. Freshness: Index lag time
5. Utilization: CPU, memory, disk usage

Alert Conditions:

- Error rate > 1% for 5 minutes
- Response time P95 > 500ms for 10 minutes
- Index lag > 5 minutes
- Cache hit rate < 60%
- Queue depth > 1000 items

Dashboard Requirements:

- Real-time search volume
- Top search queries
- Filter usage heatmap
- Performance degradation detection
- Content gap visualization

## 3.10. Security & Compliance

Access Control:

- API keys for internal services
- JWT validation for user searches
- IP whitelisting for admin operations

Data Privacy:

- Do not log personally identifiable information in search logs
- Anonymize user IDs in analytics after 30 days
- Search queries encrypted at rest

Compliance:

- GDPR: Right to be forgotten applies to search history
- CCPA: Opt-out of search analytics
- Data retention: Search logs kept for 90 days

Audit Logging:

- All admin operations (reindex, config changes)
- Access to sensitive filters (user-specific data)
- Data export/delete requests

## 3.11. Disaster Recovery

Backup Strategy:

- Daily Meilisearch index snapshots
- Point-in-time recovery capability
- Backup retention: 7 daily, 4 weekly, 12 monthly

Recovery Procedures:

Scenario: Meilisearch Outage

1. Switch to PostgreSQL fallback search (limited functionality)
2. Display banner: "Search temporarily limited"
3. Attempt automatic recovery (restart service)
4. If unavailable > 15 minutes: Restore from snapshot
5. Full recovery time objective: 30 minutes

Scenario: Data Corruption

1. Stop sync service
2. Rebuild index from PostgreSQL source
3. Validate data integrity
4. Resume sync service
5. Maximum data loss: 5 minutes

Scenario: Configuration Error

1. Rollback to previous known-good config
2. Clear cache
3. Warm up index
4. Monitor for stabilization

## 3.12. Testing Requirements

Unit Tests:

- Query parsing and transformation
- Filter syntax validation
- Cache invalidation logic

Integration Tests:

- End-to-end search flow
- Sync service data consistency
- API response formatting

Load Tests:

- Simulate 5000 concurrent users
- Mixed query patterns (simple/complex)
- Spike traffic handling

Chaos Tests:

- Meilisearch restart during peak load
- Network latency injection
- Cache failure scenarios

Acceptance Criteria:

- Search returns relevant results
- Filters work correctly in combination
- Performance meets SLAs under load
- Recovery procedures work as specified

# 4: UI/UX SPECIFICATIONS

```
markdown
## 4. UI/UX SPECIFICATIONS

### 4.1. Design System Foundation

**Design Tokens Specification:**

**Color Palette:**
- **Primary:** Indigo-500 (#6366f1) for main actions
- **Secondary:** Emerald-500 (#10b981) for success/432Hz
- **Error:** Red-500 (#ef4444) for errors/warnings
- **Warning:** Amber-500 (#f59e0b) for warnings
- **Info:** Blue-500 (#3b82f6) for information
- **Backgrounds:**
  - Light: White (#ffffff), Gray-50 (#f9fafb)
  - Dark: Gray-900 (#111827), Gray-800 (#1f2937)

**Frequency-Specific Colors:**
- 440Hz: Purple-500 (#8b5cf6)
- 432Hz: Emerald-500 (#10b981)
```

- 528Hz: Sky-500 (#0ea5e9)

**Typography Scale:**
- **Font Family:** Inter (system font stack fallback)
- **Base Size:** 16px (1rem)
- **Scale:**
  - xs: 0.75rem (12px)
  - sm: 0.875rem (14px)
  - base: 1rem (16px)
  - lg: 1.125rem (18px)
  - xl: 1.25rem (20px)
  - 2xl: 1.5rem (24px)
  - 3xl: 1.875rem (30px)

**Spacing System:**
- **Base Unit:** 4px (0.25rem)
- **Scale:** 1, 2, 3, 4, 6, 8, 10, 12, 16, 20, 24, 32, 40, 48, 56, 64

**Border Radius:**
- **sm:** 4px (0.25rem)
- **md:** 8px (0.5rem)
- **lg:** 12px (0.75rem)
- **xl:** 16px (1rem)
- **full:** 9999px

**Shadows:**
- **sm:** 0 1px 2px 0 rgb(0 0 0 / 0.05)
- **md:** 0 4px 6px -1px rgb(0 0 0 / 0.1)
- **lg:** 0 10px 15px -3px rgb(0 0 0 / 0.1)
- **xl:** 0 20px 25px -5px rgb(0 0 0 / 0.1)

(Not: Başarısız olan işlemler için 'Dead Letter Queue' mekanizması kurulacak ve Admin paneline raporlanacaktır.)

### 4.2. Core Component Specifications

**Audio Player Component:**

**States:**
1. **Loading:** Skeleton waveform, disabled controls
2. **Ready:** Play button enabled, waveform loaded
3. **Playing:** Animated progress, pause button visible

4. **Paused:** Static progress, play button visible
5. **Error:** Error message, retry button
6. **Buffering:** Loading indicator over progress bar

**Controls:**
- **Play/Pause:** Toggle with smooth transition
- **Progress Bar:** Draggable, shows buffered segments
- **Volume:** Slider with mute toggle
- **Speed Control:** 0.75x, 1x, 1.25x, 1.5x
- **Skip:** Forward/backward 10 seconds
- **Loop:** Toggle loop mode
- **Frequency Toggle:** 440Hz ↔ 432Hz (smooth crossfade)

**Visual Feedback:**
- **Current Time:** MM:SS format, updates in real-time
- **Duration:** MM:SS format
- **Buffer Progress:** Light gray fill behind progress
- **Frequency Badge:** Colored indicator (purple/green)
- **Quality Badge:** AAC/FLAC indicator

**Keyboard Shortcuts:**
- **Space:** Play/Pause
- **→:** Forward 10s
- **←:** Backward 10s
- **F:** Toggle favorite
- **M:** Mute/Unmute
- **L:** Toggle loop
- **↑/↓:** Volume control

---

### 4.3. Search Interface Specifications

**Search Bar Component:**

**States:**
1. **Idle:** Placeholder "Search for tracks, genres, moods..."
2. **Typing:** Real-time suggestions dropdown
3. **Loading:** Spinner in search button
4. **Results:** Clear button appears, suggestions hide

**Features:**

- **Debounced Search:** 300ms delay after typing stops
- **Search Suggestions:** Based on popular searches and user history
- **Recent Searches:** Last 5 searches with clear option
- **Voice Search:** Microphone icon (optional)

**Visual Query Builder:**

**Interface Elements:**
- **Filter Chips:** Add/remove filters as visual tags
- **Operator Selector:** AND/OR/NOT between filters
- **Value Inputs:** Context-aware inputs (sliders, dropdowns, text)
- **Grouping:** Parentheses for complex logic

**Example Workflow:**

[Add Filter] → Select "Genre" → Select "Acoustic" → [AND] → [Add Filter] → Select "BPM" → Drag slider 90-120 → [AND NOT] →
[Add Filter] → Select "Instrument" → Select "Drums"

```
text
**Saved Searches:**
- **Save Button:** Appears after applying filters
- **Naming Modal:** Enter custom name, notification toggle
- **Management:** List view with edit/delete/run options
- **Notifications:** Badge when new tracks match criteria

---

### 4.4. Track Card Component

**Layout (Grid Item):**
```

[Thumbnail] [Title]
[Metadata] [Actions]

```
text
```

**Thumbnail:**
- **Size:** 120px × 120px
- **Content:** Album art or generated waveform
- **Overlay:** Play button on hover
- **Badge:** Frequency indicator (corner)

**Metadata Section:**
- **Title:** Truncated after 2 lines, full on hover
- **Stats:** BPM, Duration, Key
- **Tags:** Genre, Mood (max 3, more with "+N")
- **Quality:** Format badges (AAC, FLAC)

**Action Buttons:**
- **Play/Pause:** Quick preview (30s snippet)
- **Like/Unlike:** Heart icon with count
- **Add to Queue:** Plus icon
- **More Options:** Context menu (kebab icon)

**States:**
- **Default:** All controls visible
- **Playing:** Highlight border, animated play indicator
- **Selected:** Blue border for bulk actions
- **Disabled:** Grayed out for unavailable tracks

**Hover Effects:**
- Scale up: 1.02 transform
- Shadow elevation increase
- Action buttons fade in
- Preview starts after 500ms hover

---

### 4.5. Player Interface Specifications

**Now Playing Panel:**

**Layout:**

[Album Art Large] [Track Info] [Visualizer]
[Progress Bar] [Controls] [Queue Toggle]

```
text
**Album Art:**
- **Size:** 300px × 300px (desktop), 200px × 200px (mobile)
- **Animation:** Slow rotation (360° in 60s) when playing
- **Interaction:** Click to expand fullscreen view

**Track Info Section:**
- **Title:** Large, bold
- **Artist/Composer:** "AuraStream AI" by default
- **Metadata Grid:** BPM, Key, Duration, Frequency
- **Tags:** Interactive, click to search similar

**Visualizer Options:**
1. **Waveform:** Traditional amplitude display
2. **Frequency Bars:** Real-time FFT visualization
3. **Particle Field:** Music-reactive particles
4. **Circular Spectrum:** Circular frequency display

**Mini-Player (Sticky):**
- **Height:** 64px fixed at bottom
- **Content:** Small album art, truncated title, basic
controls
- **Expand:** Click to open full player
- **Collapse:** Swipe down (mobile), click away (desktop)

---

### 4.6. License Wizard Flow

**Step 1: Usage Type Selection**

**UI Elements:**
- **Card Grid:** 6 options (YouTube, Podcast, Ad, etc.)
- **Selection:** Single select with visual feedback
- **Guidance:** Description and usage limits for each type
- **Validation:** Must select before proceeding

**Step 2: Project Details**

**Form Fields:**
- **Project Name:** Required, min 3 chars, max 100
- **YouTube Channel ID:** Optional, validation pattern
```

- **Estimated Views:** Radio buttons (0-10K, 10K-100K, etc.)
- **Format Selection:** Checkbox group with pricing
- **Frequency Selection:** Radio buttons with upsell options

**Real-time Pricing:**
- **Base Price:** $9.00 (MP3, 440Hz, single use)
- **Add-ons:**
  - FLAC: +$5.00
  - 432Hz: +$2.00
  - WAV: +$10.00
  - Stems: +$30.00
- **Total Display:** Live update as selections change

**Step 3: Review & Payment**

**Order Summary:**
- Track details with preview
- Selected options with prices
- Total amount
- License terms summary

**Legal Requirements:**
- **Checkbox 1:** "I agree to Terms of Service"
- **Checkbox 2:** "I understand AuraStream is not responsible for 3rd-party platform changes"
- **Checkbox 3:** "I will use Dispute Support Pack for Content ID claims"

**Payment Flow:**
1. **Payment Method:** Card, PayPal, Crypto options
2. **Processing:** Loading state with estimated time
3. **Success:** Download starts automatically, email sent
4. **Error:** Clear error message, retry options

---

### 4.7. Admin QC Interface Specifications

**Dual-Panel Layout:**

**Left Panel (Track List):**

- **Filter Bar:** Status, genre, date range
- **List View:** Compact track cards with status badges
- **Sorting:** By date, priority, or assignment
- **Batch Actions:** Select multiple for bulk operations

**Right Panel (QC Station):**

**Waveform Editor:**
- **Zoom:** Horizontal scroll/zoom controls
- **Selection:** Click+drag to select region
- **Trim Handles:** Draggable start/end markers
- **Playhead:** Real-time position indicator

**Metadata Editor:**
- **Form Fields:** Title, genre, mood tags, instruments
- **Auto-suggest:** Based on existing catalog
- **Validation:** Required fields highlighted
- **AI Suggestions:** Pre-filled based on audio analysis

**Quality Indicators:**
- **LUFS Meter:** Visual indicator (-14 target)
- **Clipping Detection:** Red highlights on waveform
- **Spectrogram:** Toggle overlay for frequency analysis
- **Phase Meter:** Stereo phase correlation indicator

**Decision Buttons:**
- **Approve:** Green button, moves to processing
- **Reject:** Red button with reason dropdown
- **Request Changes:** Yellow button with notes field
- **Skip:** Gray button, moves to next

**Keyboard Shortcuts (QC):**
- **A:** Approve
- **R:** Reject
- **S:** Skip
- **Space:** Play/Pause
- **←/→:** Nudge trim markers
- **Ctrl+Z:** Undo last action

---

### 4.8. Loading States Specification

**Skeleton Screens:**

**Search Results:**
- 8 track card skeletons
- Animated gradient overlay
- Progressive loading (2 batches)

**Player:**
- Waveform skeleton (animated bars)
- Metadata lines (varying widths)
- Control buttons (circles)

**Dashboard:**
- Card skeletons with charts
- Table rows with placeholder text
- Stat cards with loading numbers

**Loading Indicators:**

**Size Variants:**
- **Small:** 16px spinner for buttons
- **Medium:** 32px spinner for sections
- **Large:** 64px spinner for full page

**Type Variants:**
- **Spinner:** Circular rotation
- **Progress Bar:** Determinate for known duration
- **Skeleton:** Content placeholder
- **Pulse:** Subtle background animation

**Loading Messages:**
- **Audio:** "Loading audio... (35%)"
- **Search:** "Finding matching tracks..."
- **Processing:** "Generating 432Hz version..."
- **Payment:** "Securing your license..."

---

### 4.9. Error States & Recovery

**Error Types & UI Responses:**

**Network Errors:**

- **Detection:** Failed API calls, offline detection
- **UI:** Red banner "Connection lost"
- **Actions:** "Retry", "Work Offline"
- **Fallback:** Show cached content if available

**Authentication Errors:**
- **Detection:** 401/403 responses
- **UI:** Modal "Session expired"
- **Actions:** "Re-login", "Continue as guest"
- **Redirect:** To login page with return URL

**Payment Errors:**
- **Detection:** Stripe/charge failures
- **UI:** Inline error below payment form
- **Actions:** "Try different card", "Contact support"
- **Preservation:** Save cart, auto-retry with backup method

**Media Playback Errors:**
- **Detection:** Audio load/play failures
- **UI:** Overlay on player "Could not play track"
- **Actions:** "Try again", "Next track", "Report issue"
- **Fallback:** Switch to lower quality format

**Search Errors:**
- **Detection:** Timeout or empty results
- **UI:** "No results found" with suggestions
- **Actions:** "Broaden search", "Try similar terms"
- **Fallback:** Show popular tracks in same genre

**Error Messaging Guidelines:**
- **User-friendly:** No technical jargon
- **Actionable:** Clear next steps
- **Specific:** Include relevant details
- **Consistent:** Same error, same message

---

### 4.10. Mobile-First Responsive Design

**Breakpoints:**
- **Mobile:** 320px - 767px (single column)
- **Tablet:** 768px - 1023px (two column)

- **Desktop:** 1024px+ (three column)

**Mobile Specific Patterns:**

**Navigation:**
- **Bottom Tabs:** Player, Search, Library, Profile
- **Swipe Gestures:**
  - Left/right: Navigate tabs
  - Down: Close modals
  - Up: Refresh content
- **Large Touch Targets:** Minimum 44px × 44px

**Player Adaptation:**
- **Collapsed:** Fixed mini-player at bottom
- **Expanded:** Fullscreen modal
- **Controls:** Larger buttons, simplified layout
- **Gestures:** Swipe left/right to change tracks

**Tablet Adaptation:**
- **Split View:** List on left, detail on right
- **Floating Player:** Bottom-right corner
- **Multi-touch:** Supported for advanced controls

**Desktop Enhancements:**
- **Multi-panel:** Search filters, results, queue visible
- **Keyboard Navigation:** Full support
- **Hover States:** Rich interactions
- **Window Management:** Resizable panels

---

### 4.11. Accessibility Requirements

**WCAG 2.1 AA Compliance:**

**Screen Reader Support:**
- **ARIA Labels:** All interactive elements
- **Live Regions:** Dynamic updates announced
- **Landmarks:** Proper page structure
- **Focus Management:** Logical tab order

**Keyboard Navigation:**
- **Focus Indicators:** Visible focus rings

- **Skip Links:** Jump to main content
- **Shortcuts:** Documented and consistent
- **Escape:** Close all modals/dropdowns

**Visual Accessibility:**
- **Color Contrast:** Minimum 4.5:1 for text
- **Text Scaling:** Support up to 200%
- **Reduced Motion:** Respect OS preference
- **High Contrast Mode:** Supported

**Audio Accessibility:**
- **Volume Control:** Independent of system
- **Visual Alternatives:** Waveform/spectrum displays
- **Captioning:** For tutorial videos
- **Audio Descriptions:** Where applicable

**Testing Requirements:**
- **Automated:** axe-core integration in CI
- **Manual:** Screen reader testing (NVDA, VoiceOver)
- **User Testing:** Include users with disabilities
- **Compliance:** Quarterly accessibility audit

---

### 4.12. Animation & Micro-interactions

**Performance Guidelines:**
- **60fps Target:** All animations smooth
- **GPU Acceleration:** Use transform/opacity
- **Debounce:** Prevent animation jank
- **Fallback:** Graceful degradation

**Standard Animations:**

**Transitions:**
- **Fade:** 150ms ease-in-out
- **Slide:** 200ms ease-out
- **Scale:** 100ms ease-in (buttons)
- **Crossfade:** 300ms for audio transitions

**Loading Animations:**
- **Skeleton Pulse:** 2s infinite
- **Spinner Rotation:** 1s linear infinite

- **Progress Fill:** Ease-out based on actual progress

**Feedback Animations:**
- **Button Press:** Scale down 0.95
- **Success Checkmark:** Scale and fade
- **Error Shake:** Horizontal 5px shake
- **Hover Lift:** TranslateY(-2px) with shadow

**Audio-Specific Animations:**
- **Visualizer:** Smooth FFT updates
- **Progress Bar:** Real-time smooth movement
- **Frequency Switch:** 3s crossfade with color transition
- **Beat Detection:** Subtple pulse on beats

**Reduced Motion:**
- **Detection:** `prefers-reduced-motion`
- **Fallback:** Remove non-essential animations
- **Essential:** Keep progress indicators
- **Testing:** Verify with motion sensitivity

---

### 4.13. User Onboarding & Guidance

**First-Time Experience:**

**Progressive Disclosure:**
1. **App Install:** PWA installation prompt
2. **First Launch:** Quick feature tour (3 screens)
3. **First Search:** Tooltip on search filters
4. **First Play:** Player controls highlight
5. **First License:** Step-by-step wizard

**Empty States:**
- **No Searches:** "Try searching for..."
- **No Favorites:** "Like tracks to save them here"
- **No Downloads:** "Your licensed tracks will appear here"
- **No History:** "Played tracks will appear here"

**Tooltips & Help:**
- **Contextual:** Appear on first interaction
- **Persistent:** "?" icons for complex features
- **Dismissible:** User can hide after viewing

- **Rediscoverable:** Accessible via help menu

**Tutorial System:**
- **Video Tutorials:** 1-2 minute explainers
- **Interactive Guides:** Step-by-step walkthroughs
- **Cheat Sheets:** Keyboard shortcuts, tips
- **Context Help:** "Learn more" links throughout

---

### 4.14. Performance Budget

**Core Web Vitals Targets:**

**Largest Contentful Paint (LCP):**
- Target: < 2.5 seconds
- Measurement: First meaningful paint
- Optimization: Critical CSS, image optimization

**First Input Delay (FID):**
- Target: < 100 milliseconds
- Measurement: First interaction responsiveness
- Optimization: Code splitting, minimal JS

**Cumulative Layout Shift (CLS):**
- Target: < 0.1
- Measurement: Visual stability
- Optimization: Reserve space, stable fonts

**Additional Metrics:**

**Time to Interactive (TTI):**
- Target: < 3.5 seconds
- Critical Path: API calls, component hydration

**Bundle Size Limits:**
- **Initial Load:** < 200KB gzipped
- **Total JS:** < 500KB per app
- **Chunk Strategy:** Route-based code splitting

**Asset Optimization:**
- **Images:** WebP format, responsive srcset
- **Audio:** Lazy loading, progressive quality

- **Fonts:** WOFF2, subsetting, font-display: swap

**Caching Strategy:**
- **Service Worker:** Offline support for core features
- **CDN:** All static assets cached
- **API:** Stale-while-revalidate for metadata
- **Local Storage:** User preferences, recent data

---

### 4.15. Cross-Browser Compatibility

**Supported Browsers:**

**Desktop:**
- Chrome 90+ (primary)
- Firefox 88+
- Safari 14+
- Edge 90+

**Mobile:**
- iOS Safari 14+
- Chrome for Android 90+
- Samsung Internet 14+

**Feature Detection & Fallbacks:**

**Web Audio API:**
- **Primary:** Web Audio API for advanced features
- **Fallback:** HTML5 Audio for basic playback
- **Detection:** `window.AudioContext` check

**PWA Features:**
- **Install Prompt:** Chrome, Firefox, Edge
- **Fallback:** Add to home screen instructions
- **Offline:** Service worker with cache-first strategy

**CSS Features:**
- **Grid/Flexbox:** Modern layout techniques
- **Fallback:** Float-based layouts for old browsers
- **Prefixing:** Autoprefixer in build process

**Testing Matrix:**

- **Automated:** BrowserStack for visual regression
- **Manual:** Critical user flows on all browsers
- **Priority:** Mobile Safari (largest user base)
- **Monitoring:** Real User Monitoring (RUM) data

# 5: OPERATIONAL EXCELLENCE & DEVOPS

markdown
## 5. OPERATIONAL EXCELLENCE & DEVOPS

### 5.1. Performance Monitoring Strategy

**Monitoring Stack Architecture:**

**Application Performance Monitoring (APM):**
- **Primary:** Sentry (error tracking + performance)
- **Secondary:** Vercel Analytics (frontend metrics)
- **Infrastructure:** AWS CloudWatch + Custom metrics

**Key Performance Indicators (KPIs):**

**Frontend Performance:**
- **LCP (Largest Contentful Paint):** < 2.5s (P95)
- **FID (First Input Delay):** < 100ms (P95)
- **CLS (Cumulative Layout Shift):** < 0.1
- **TTI (Time to Interactive):** < 3.5s

**Backend Performance:**
- **API Response Time:** < 200ms (P95) for search, < 100ms for streaming
- **Database Query Time:** < 50ms (P95) for simple queries
- **Cache Hit Rate:** > 80% for search results
- **Error Rate:** < 0.1% of total requests

**Business Metrics:**
- **Active Venues:** Daily count with playback > 1 hour
- **Creator Engagement:** Searches per user, licenses per month
- **Content Velocity:** Tracks processed per day, approval rate
- **Revenue Metrics:** MRR, churn rate, LTV

**Real User Monitoring (RUM):**
- **Sample Rate:** 100% for errors, 10% for performance
- **Data Collected:** Browser, device, location, connection type
- **Anonymization:** IP masking, no PII collection
- **Retention:** 90 days raw, 1 year aggregated

---

### 5.2. Error Tracking & Logging Specification

**Logging Standards:**

**Log Levels:**
- **ERROR:** System failures, data corruption, security issues
- **WARN:** Deprecations, rate limiting, degraded performance
- **INFO:** Business events (license created, track approved)
- **DEBUG:** Development debugging (disabled in production)
- **TRACE:** Detailed request/response tracing

**Log Structure (JSON Format):**
```json
{
  "timestamp": "2024-01-15T10:30:00.000Z",
  "level": "ERROR",
  "service": "search-api",
  "environment": "production",
  "release": "v1.2.3",
  "message": "Meilisearch connection timeout",
  "error": {
    "type": "ConnectionError",
    "code": "ETIMEDOUT",
```

```
    "stack": "..."
  },
  "context": {
    "request_id": "req_123",
    "user_id": "usr_456",
    "endpoint": "/api/v1/search",
    "params": {"q": "acoustic"}
  },
  "metadata": {
    "duration_ms": 2500,
    "retry_count": 3
  }
}
```

Error Classification:

Severity Levels:

- P0 (Critical): Service down, data loss, security breach
  - Response: Immediate page, fix within 1 hour
- P1 (High): Major feature broken, performance degradation
  - Response: Address within 4 hours
- P2 (Medium): Minor feature issues, non-critical errors
  - Response: Address within 24 hours
- P3 (Low): Cosmetic issues, edge cases
  - Response: Address in next release

Error Budget Policy:

- Monthly Allowance: 0.1% error rate
- Burn Rate Alerts:
  - 2-hour burn > 10%: Warning
  - 1-hour burn > 20%: Critical
- Consequences: Feature freeze if budget exhausted

Alerting Rules:

Immediate Alerts (PagerDuty/SMS):

- Service unavailable > 5 minutes
- Database replication lag > 30 seconds
- Payment processing errors > 1%
- Security event detected

Daily Digest (Email/Slack):

- Error rate trends
- Performance degradation
- Usage anomalies
- Cost overruns

Weekly Report:

- Error budget status
- Mean time to detection/resolution
- Top recurring issues
- Improvement recommendations

## 5.3. Backup & Disaster Recovery (DR)

Data Classification & Retention:

Critical Data (Zero data loss tolerance):

- User accounts and profiles
- License records
- Payment transactions
- Backup: Real-time replication + hourly snapshots
- Retention: 7 days hourly, 30 days daily, 1 year monthly

Important Data (Minimal loss acceptable):

- Track metadata
- Playback analytics
- Search logs
- Backup: Daily snapshots + point-in-time recovery

- Retention: 30 days daily, 1 year weekly

Transient Data (Recreatable):

- Cache contents
- Session data
- Temporary files
- Backup: None (regenerate on recovery)
- Retention: Live only

Backup Implementation:

Database (Supabase):

- Automated: Point-in-Time Recovery (PITR) enabled
- Frequency: Continuous WAL archiving
- Retention: 7 days of continuous recovery
- Testing: Monthly restore drill

File Storage (S3):

- Versioning: Enabled on all buckets
- Replication: Cross-region for critical buckets
- Lifecycle:
  - Current versions: Keep forever
  - Previous versions: 30 days then archive
  - Delete markers: 7 days then permanent delete

Application State:

- Configuration: Git repository with secrets management
- Database Schema: Migration files in version control
- Search Index: Rebuild from source (Supabase)

Disaster Recovery Scenarios:

Scenario A: Database Corruption

- RTO (Recovery Time Objective): 15 minutes
- RPO (Recovery Point Objective): 5 minutes
- Procedure:

1. Isolate affected database
2. Restore from latest clean snapshot
3. Apply WAL logs up to corruption point
4. Validate data integrity
5. Redirect traffic

Scenario B: Region Outage (AWS eu-central-1)

- RTO: 30 minutes
- RPO: 15 minutes
- Procedure:

1. Failover DNS to backup region
2. Promote read replicas to primary
3. Update application configuration
4. Monitor performance in new region

Scenario C: Data Center Loss

- RTO: 2 hours
- RPO: 1 hour
- Procedure:

1. Activate cold standby in alternate provider
2. Restore from offsite backups
3. Update DNS globally
4. Communicate status to users

DR Testing Schedule:

- Monthly: Backup restoration test
- Quarterly: Failover simulation
- Biannually: Full DR drill
- Annually: Third-party audit

## 5.4. CDN & Cache Strategy

Cache Hierarchy:

Layer 1: Browser Cache

- Assets: CSS, JS, fonts, icons
- TTL: 1 year (immutable with hash-based filenames)
- Invalidation: New build = new filename

Layer 2: CDN Edge (CloudFront)

- Static Assets: Images, audio previews, PDFs
- TTL: 30 days for images, 1 year for audio files
- Invalidation: Per-file (not wildcard) to control cost

Layer 3: Application Cache (Redis)

- API Responses: Search results, user data, track metadata
- TTL: 5 minutes for dynamic data, 1 hour for semi-static
- Strategy: Stale-while-revalidate

Layer 4: Database Cache (PostgreSQL)

- Query Results: Materialized views for analytics
- TTL: Refreshed hourly/daily based on freshness needs
- Management: Automated refresh during low traffic

Cache Invalidation Rules:

Event-Based Invalidation:

- Track Updated: Invalidate search cache for track genre/mood
- New Track Added: Invalidate "newest" queries and related facets
- License Created: Update user's license cache
- User Data Changed: Invalidate user-specific caches

Time-Based Invalidation:

- Popular Content: Shorter TTL (1-5 minutes)
- Static Content: Longer TTL (days/weeks)
- Analytics Data: Refreshed on schedule (hourly/daily)

Cache Key Design:

```
text
{service}:{resource}:{identifier}:{version}:{user_context}
```

```
Examples:
- search:tracks:acoustic:90-120:chill:v2:premium_user
- audio:stream:track_123:aac:432hz:v1
- user:licenses:usr_456:active:v1
```

Performance Targets:

- Cache Hit Rate: > 80% overall
- Cache Response Time: < 10ms (Redis), < 50ms (CDN)
- Cache Miss Penalty: < 200ms additional latency
- Cache Size Management: LRU eviction with monitoring

## 5.5. API Rate Limiting Rules

Rate Limiting Architecture:

Implementation Layer:

- Edge: CloudFront + WAF for DDoS protection
- Application: Redis-based sliding window algorithm
- Service: Per-service limits (Suno API, Stripe, etc.)

Limiting Algorithms:

- Sliding Window: For precise minute/hour limits
- Token Bucket: For burst allowance
- Fixed Window: For simple daily/monthly limits

Rate Limit Tiers:

Tier 1: Anonymous Users

- Limit: 20 requests per minute per IP
- Burst: 30 requests (50% allowance)
- Scope: All public endpoints
- Purpose: DDoS protection, resource conservation

Tier 2: Authenticated Creators

- Limit: 60 requests per minute per user
- Burst: 90 requests
- Scope: All endpoints except streaming
- Purpose: Fair usage, prevent abuse

Tier 3: Venue Players

- Limit: 1000 requests per minute per venue
- Burst: 1500 requests
- Scope: Streaming, playlist sync, heartbeats
- Purpose: Support continuous playback, sync operations

Tier 4: Admin Users

- Limit: 5000 requests per minute per admin
- Burst: 7500 requests
- Scope: All endpoints
- Purpose: Administrative operations, bulk actions

Special Endpoint Limits:

Search API:

- Base: 100 requests/minute
- Complex queries (multiple filters): 30 requests/minute
- Faceted searches: 50 requests/minute

Streaming API:

- Base: 1000 requests/minute (venue tier)
- Per-track limit: 100 requests/minute (prevent hotlinking)

License Generation:

- Base: 10 licenses/minute per user
- Fraud prevention: Velocity checks (geographic, time-based)

Response Headers:

```
text
X-RateLimit-Limit: 60
X-RateLimit-Remaining: 45
X-RateLimit-Reset: 1610737200
```

```
Retry-After: 30
```

Rate Limit Exceeded Response:

```
json
{
  "error": {
    "code": "rate_limit_exceeded",
    "message": "Too many requests. Please try again in 30
seconds.",
    "retry_after": 30,
    "limits": {
      "max": 60,
      "remaining": 0,
      "reset": 1610737200
    }
  }
}
```

Monitoring & Adjustments:

- Weekly Review: Limit effectiveness, false positives
- Auto-adjustment: Based on user tier changes
- Emergency Override: Admin capability for legitimate cases
- Metrics: Rejection rate, top limited users, endpoint patterns

## 5.6. Security Monitoring & Compliance

Security Monitoring Stack:

Vulnerability Scanning:

- Dependencies: Weekly Snyk/GitHub Dependabot scans
- Containers: Trivy scanning in CI/CD pipeline
- Infrastructure: AWS Inspector monthly scans

- Web Application: OWASP ZAP automated scanning

Intrusion Detection:

- Network: AWS GuardDuty for anomalous behavior
- Application: Custom rules for suspicious patterns
- User Behavior: Login anomalies, geographic jumps
- Data Access: Unusual query patterns, bulk exports

Compliance Monitoring:

GDPR Compliance:

- Data Mapping: All PII identified and cataloged
- Consent Tracking: User preferences logged and versioned
- Right to Erasure: Automated process with verification
- Data Portability: Export functionality with audit trail

PCI DSS Requirements:

- Card Data: Never stored (Stripe Elements only)
- Logging: No PAN, CVV, or track data in logs
- Access Control: Segregated duties for payment systems
- Network Security: TLS 1.2+, WAF protection

Security Incident Response:

Classification:

- Severity 1: Active breach, data exfiltration
- Severity 2: Vulnerability with exploit available
- Severity 3: Security misconfiguration
- Severity 4: Informational findings

Response Times:

- Sev 1: Immediate (within 1 hour)
- Sev 2: 4 business hours
- Sev 3: 3 business days
- Sev 4: Next release cycle

Forensic Requirements:

- Log Retention: 1 year for security-relevant logs
- Audit Trail: Immutable logging for critical operations
- Chain of Custody: Document evidence handling
- Reporting: Incident reports with lessons learned

Security Training & Awareness:

- Quarterly: Security training for engineering team
- Annual: Phishing simulation exercises
- Continuous: Security newsletter with updates
- Onboarding: Security training for new hires

## 5.7. Cost Management & Optimization

Cost Allocation & Tagging:

AWS Resource Tagging:

- `Environment`: production, staging, development
- `Project`: aurastream
- `Component`: api, database, storage, search
- `CostCenter`: engineering, marketing, operations
- `Owner`: team/individual responsible

Monthly Budgets:

Infrastructure Budget:

- Compute (Lambda, EC2): $500/month
- Storage (S3, EBS): $200/month
- Data Transfer (CloudFront): $300/month
- Database (RDS): $400/month
- Search (Meilisearch): $100/month

Service Budgets:

- Suno AI API: $500/month (hard limit)
- Stripe Fees: Variable (2.9% + $0.30)
- Monitoring (Sentry): $50/month

- Email/SMS: $100/month

Cost Optimization Strategies:

Compute Optimization:

- Lambda: Right-sizing memory, provisioned concurrency
- Auto-scaling: Scale in during off-peak hours
- Spot Instances: For non-critical batch jobs
- Reserved Instances: For predictable baseline load

Storage Optimization:

- S3 Lifecycle: Move to Infrequent Access after 30 days
- Data Compression: GZIP for logs, Brotli for web assets
- Deduplication: Identify and eliminate duplicate files
- Cleanup Automation: Remove temporary/unused data

Data Transfer Optimization:

- CDN Optimization: Regional edge caching
- Compression: Enable compression for all text-based responses
- Connection Reuse: HTTP/2, keep-alive connections
- Lazy Loading: Load non-critical assets on demand

Cost Monitoring & Alerts:

Alert Thresholds:

- Daily: > 150% of expected daily burn
- Weekly: > 120% of weekly budget
- Monthly: > 100% of monthly budget
- Anomaly: Unusual cost spike (> 3x normal)

Cost Reporting:

- Daily: Top 10 cost drivers
- Weekly: Budget vs actual, forecast
- Monthly: Cost per feature, ROI analysis
- Quarterly: Optimization opportunities review

FinOps Practices:

- Showback: Teams see their infrastructure costs
- Chargeback: Business units accountable for costs
- Optimization Champions: Each team has cost responsibility
- Cost-aware Development: Cost considerations in design reviews

## 5.8. Deployment Pipeline Specification

CI/CD Pipeline Stages:

Stage 1: Pre-commit Validation

- Tools: Husky git hooks
- Checks:
  - ESLint (code style)
  - TypeScript compilation
  - Unit tests (affected files only)
  - Secrets detection
- Requirements: All checks must pass to commit

Stage 2: Pull Request Validation

- Environment: Dedicated preview deployment
- Tests:
  - Full test suite (unit + integration)
  - E2E tests (critical paths)
  - Performance regression tests
  - Security vulnerability scan
- Requirements:
  - 2+ approvals required
  - All tests green
  - No security vulnerabilities (critical/high)

Stage 3: Staging Deployment

- Environment: production-like staging
- Process:
    - Database migration (if any)
    - Application deployment
    - Cache warm-up
    - Health checks
- Validation:
    - Smoke tests (5 critical user journeys)
    - Performance benchmark
    - Canary testing (10% traffic)
    - Monitoring for anomalies (1 hour)

Stage 4: Production Deployment

- Strategy: Blue-green deployment
- Process:
    1. Deploy to blue environment
    2. Run health checks
    3. Switch 10% traffic
    4. Monitor for 15 minutes
    5. Gradually increase to 100%
    6. Decommission old environment
- Rollback: Automatic if error rate > 2%

Deployment Windows:

- Low-risk changes: Anytime (automated)
- Medium-risk changes: Business hours (10AM-2PM)
- High-risk changes: Maintenance window (Sunday 2AM-4AM)
- Emergency fixes: Anytime with on-call approval

Infrastructure as Code:

- AWS: Terraform for all resources
- Kubernetes: Helm charts (if applicable)
- Database: Migration files versioned in Git
- Configuration: Environment-specific config files

Deployment Metrics:

- Deployment Frequency: Target: daily, Minimum: weekly
- Lead Time: Code commit to production < 1 hour
- Change Failure Rate: < 5%
- Mean Time to Recovery: < 30 minutes

## 5.9. Capacity Planning & Scaling

Capacity Metrics:

Current Scale (Launch):

- Users: 1,000 concurrent maximum
- Tracks: 10,000 active tracks
- Storage: 500GB audio files
- Bandwidth: 1TB/month

Growth Projections:

- Month 1-3: 100% month-over-month growth
- Month 4-6: 50% month-over-month growth
- Month 7-12: 25% month-over-month growth
- Year 2: 10% month-over-month growth

Scaling Triggers:

Horizontal Scaling (Add more instances):

- CPU Utilization: > 70% for 5 minutes
- Memory Utilization: > 80% for 5 minutes
- Queue Depth: > 1000 pending messages
- Response Time: P95 > 300ms for 10 minutes

Vertical Scaling (Increase instance size):

- Consistent: > 60% utilization for 7 days
- Predictable Growth: Linear growth trend identified
- Cost Optimization: Larger instances more cost-effective

Auto-scaling Rules:

Application Servers:

- Scale Out: CPU > 70% OR memory > 75%
- Scale In: CPU < 30% AND memory < 40%
- Cooldown: 300 seconds between scaling actions
- Max Instances: 10 (per environment)

Database:

- Read Replicas: Add when read CPU > 60%
- Storage: Auto-extend with 20% buffer
- Performance Insights: Enable for query optimization

Search Engine:

- Memory: Scale when > 80% utilization
- Disk: Alert when < 20% free space
- Index Size: Split when > 1 million documents

Load Testing Requirements:

- Monthly: Full system load test
- Pre-release: Feature-specific load tests
- Capacity Validation: Before major marketing campaigns
- Disaster Recovery: Load test after infrastructure changes

## 5.10. Incident Management & On-call

On-call Rotation:

Schedule:

- Primary: 1-week rotation
- Secondary: Backup for primary
- Escalation: Engineering manager after 30 minutes
- Weekend Coverage: Reduced team (2 engineers)

Responsibilities:

- Monitoring: Respond to alerts within SLA
- Incident Management: Lead response during incidents
- Communication: Update status page, notify stakeholders
- Post-mortem: Document incident and lessons learned

Tooling:

- Alerting: PagerDuty for critical alerts
- Communication: Slack for team coordination
- Documentation: Runbooks in Notion/Confluence
- Status Page: Updates for external communication

Incident Severity Levels:

Sev 1 - Critical:

- Impact: Service completely down
- Response Time: 5 minutes
- Resolution Time: 1 hour
- Communication: 15-minute updates

Sev 2 - Major:

- Impact: Major feature degradation
- Response Time: 15 minutes
- Resolution Time: 4 hours
- Communication: Hourly updates

Sev 3 - Minor:

- Impact: Minor issues, workarounds available
- Response Time: 1 business day
- Resolution Time: 3 business days
- Communication: Daily updates

Sev 4 - Low:

- Impact: Cosmetic, informational
- Response Time: 1 week
- Resolution Time: Next release
- Communication: As needed

Post-Incident Process:

Within 24 hours:

- Initial incident summary
- Timeline of events
- Impact assessment

Within 3 business days:

- Root cause analysis
- Action items identified
- Process improvements suggested

Within 1 week:

- Action items assigned
- Monitoring improvements implemented
- Runbooks updated

On-call Wellness:

- Maximum: 1 week per month on-call
- Minimum: 12 hours between shifts
- Compensation: Additional PTO or monetary compensation
- Training: Regular on-call training and simulations

# 6: SECURITY

```
markdown
## 6. SECURITY

### 6.1. Security Architecture Principles

**Defense in Depth Layers:**

**Layer 1: Network Security**
- CloudFront WAF with OWASP Core Rule Set
- VPC with security groups (least privilege)
- DDoS protection (AWS Shield Standard/Advanced)
- TLS 1.2+ enforcement for all connections
```

**Layer 2: Application Security**
- Input validation at all entry points
- Output encoding to prevent XSS
- CSRF protection for state-changing operations
- Content Security Policy (CSP) headers

**Layer 3: Data Security**
- Encryption at rest (AES-256) for all sensitive data
- Encryption in transit (TLS 1.2+)
- Key management via AWS KMS with rotation
- Client-side encryption for offline data

**Layer 4: Access Security**
- Multi-factor authentication for admin accounts
- Role-based access control (RBAC)
- Principle of least privilege
- Session management with secure tokens

**Security by Design Requirements:**
- Security requirements defined before implementation
- Threat modeling for new features
- Secure coding standards enforced
- Security testing integrated into CI/CD

---

### 6.2. Authentication & Authorization

**Authentication Flow:**

**User Authentication:**
- **Primary:** Supabase Auth (JWT-based)
- **Tokens:** Access token (15 min), Refresh token (7 days)
- **Storage:** HTTP-only cookies for web, secure storage for mobile
- **MFA:** Required for admin accounts, optional for users

**JWT Token Structure:**
```json
{
  "header": {
    "alg": "RS256",
```

```
    "typ": "JWT"
  },
  "payload": {
    "sub": "user_uuid",
    "role": "creator|venue|admin",
    "tier": "free|pro|business",
    "venue_id": "venue_uuid", // if role=venue
    "iat": 1610737200,
    "exp": 1610738100,
    "refresh_token_id": "rt_uuid"
  },
  "signature": "RS256_SIGNATURE"
}
```

Token Security Measures:

- Signature: RS256 asymmetric signing
- Validation: Audience, issuer, expiration checks
- Blacklisting: Revoked tokens stored in Redis (24h TTL)
- Rotation: New refresh token on each use
- Binding: Tokens bound to user agent fingerprint

Authorization Model:

Role Definitions:

- Anonymous: Browse catalog, search, preview tracks
- Creator: License tracks, manage library, dispute center
- Venue: Stream music, manage schedule, offline cache
- Admin: QC approval, content generation, system management
- Service: Internal service-to-service communication

Permission Matrix:

| Resource | Anonymous | Creator | Venue | Admin | Service |
|---|---|---|---|---|---|

| | | | | | |
|---|---|---|---|---|---|
| Search tracks | READ | READ | READ | READ | READ |
| Stream audio | - | - | READ | READ | READ |
| License track | - | CREATE | - | CREATE | CREATE |
| Manage library | - | CRUD | - | CRUD | - |
| Approve tracks | - | - | - | CRUD | - |
| Generate content | - | - | - | CREATE | CREATE |
| System config | - | - | - | CRUD | READ |

Attribute-Based Access Control (ABAC):

- Subscription Tier Checks: Premium features (432Hz, FLAC)
- Geographic Restrictions: Content licensing by region
- Usage Limits: Tracks licensed per project
- Time-based: License expiration enforcement

## 6.3. Data Protection & Encryption

Encryption Strategy:

Data Classification:

- Level 1 (Highly Sensitive): Payment data, encryption keys
  - Encryption: AES-256-GCM at rest and in transit
  - Access: Strictly controlled, logged, and monitored
  - Backup: Encrypted, air-gapped
- Level 2 (Sensitive): User PII, license keys, audio watermarks
  - Encryption: AES-256 at rest, TLS in transit

    - Access: Role-based with audit logging
    - Backup: Encrypted
- Level 3 (Confidential): Track metadata, analytics
    - Encryption: TLS in transit, optional at rest
    - Access: Business need-to-know
    - Backup: Standard
- Level 4 (Public): Public track streams, marketing content
    - Encryption: TLS in transit
    - Access: Public with rate limiting
    - Backup: As needed

Key Management:

- Master Keys: AWS KMS with automatic rotation (yearly)
- Data Keys: Generated per encryption operation
- Key Storage: Never in code, environment variables, or logs
- Key Rotation:
    - Master keys: Annual
    - Data keys: Per encryption session
    - Compromised keys: Immediate rotation

Client-Side Encryption (Offline Cache):

- Purpose: Protect cached audio files from extraction
- Algorithm: AES-GCM with session-derived key
- Key Derivation: From user session token + device fingerprint
- Security: Keys never stored, derived on-demand
- Recovery: Re-authentication required if session expires

Watermarking & DRM:

- Audio Watermarking: Inaudible unique identifier in each download
- PDF Watermarking: Steganographic user ID in license certificates
- Stream Protection: Signed URLs with IP binding and TTL
- Rights Management: License validation before playback

## 6.4. API Security

API Gateway Security:

- Authentication: JWT validation on all endpoints
- Rate Limiting: Tier-based (see Section 5.5)
- Request Validation: Schema validation for all inputs
- Response Sanitization: Remove sensitive data from responses

Input Validation Rules:

- SQL Injection Prevention: Parameterized queries only
- XSS Prevention: HTML encoding of all user inputs
- Path Traversal: Sanitize file paths, use allowlists
- File Uploads: Validate MIME types, scan for malware
- JSON/XML Bombs: Limit request size, depth, and complexity

API Endpoint Security Levels:

Public Endpoints (No auth required):

- `GET /api/health`
- `GET /api/tracks/search` (rate limited)
- `GET /api/tracks/{id}/preview` (30-second snippets)

Authenticated Endpoints (User JWT required):

- `GET /api/stream/{id}` (subscription tier checked)
- `POST /api/licenses` (payment processing)
- `GET /api/user/library` (user-specific data)

Admin Endpoints (Admin role required):

- `POST /api/admin/tracks/{id}/approve`
- `GET /api/admin/analytics`
- `POST /api/admin/users/{id}/suspend`

Service Endpoints (Service token required):

- `POST /api/internal/sync/search` (Meilisearch sync)

- `POST /api/internal/process/audio` (FFmpeg processing)
- `GET /api/internal/metrics` (monitoring data)

Security Headers:

```
text
Strict-Transport-Security: max-age=31536000;
includeSubDomains
X-Frame-Options: DENY
X-Content-Type-Options: nosniff
Content-Security-Policy: default-src 'self'; script-src
'self' 'unsafe-inline' https://js.stripe.com; style-src
'self' 'unsafe-inline'; img-src 'self' data: https://
*.stripe.com; font-src 'self'; connect-src 'self' https://
api.stripe.com https://*.sentry.io;
X-XSS-Protection: 1; mode=block
Referrer-Policy: strict-origin-when-cross-origin
Permissions-Policy: microphone=(), camera=(),
geolocation=()
```

## 6.5. Infrastructure Security

Cloud Security Configuration:

AWS Security:

- IAM: Least privilege, no root account usage, MFA required
- VPC: Private subnets for databases, security group isolation
- S3: Block public access, encryption enabled, versioning
- CloudFront: WAF enabled, TLS 1.2+ only, logging enabled
- RDS: Encryption at rest, automated backups, no public access

Container Security:

- Base Images: Official images only, regularly updated
- Vulnerability Scanning: In CI/CD pipeline
- Runtime Security: Read-only root filesystem, non-root user

- Secrets: Environment variables via AWS Secrets Manager

Network Security:

- DDoS Protection: AWS Shield Standard (Advanced for critical)
- WAF Rules: OWASP Top 10, custom rules for business logic
- VPN: Required for administrative access
- Monitoring: VPC Flow Logs, GuardDuty for anomalies

Secrets Management:

- Storage: AWS Secrets Manager for all secrets
- Rotation: Automatic for database, API keys, certificates
- Access: IAM roles with least privilege
- Audit: All access logged and monitored

## 6.6. Vulnerability Management

Vulnerability Assessment Schedule:

Daily:

- Dependency scanning (GitHub Dependabot, Snyk)
- Infrastructure configuration checks
- Security group rule validation

Weekly:

- Container image scanning
- Secret scanning in code repositories
- Log analysis for security events

Monthly:

- Penetration testing (automated)
- Compliance scanning (CIS benchmarks)
- User permission reviews

Quarterly:

- External penetration testing
- Security architecture review
- Third-party security assessment

Annual:

- Full security audit
- Red team exercise
- Security certification renewal

Vulnerability Response Process:

Severity Classification:

- Critical: Remote code execution, data breach
  - Response: Immediate patch, maximum 24 hours
- High: Privilege escalation, data exposure
  - Response: Patch within 7 days
- Medium: Information disclosure, DoS
  - Response: Patch within 30 days
- Low: Best practice violations
  - Response: Patch in next release cycle

Patch Management:

- Emergency Patches: Deployed immediately with rollback plan
- Scheduled Patches: Monthly maintenance window
- Dependency Updates: Automated with manual approval
- Testing: Security patches tested in staging first

Third-party Risk Management:

- Vendor Assessment: Security review before integration
- API Security: Validate third-party API security practices
- Data Sharing: Minimal data sharing with clear agreements
- Monitoring: Third-party service availability and security

## 6.7. Incident Response & Forensics

Security Incident Classification:

Category A: Data Breach

- Unauthorized access to sensitive data
- Response: Immediate containment, notification procedures

Category B: Service Compromise

- Unauthorized access to systems
- Response: Isolate affected systems, investigate extent

Category C: Availability Attack

- DDoS, ransomware, sabotage
- Response: Mitigate attack, restore services

Category D: Policy Violation

- Insider threat, policy breach
- Response: Investigate, disciplinary action

Incident Response Team (IRT):

Roles:

- Incident Commander: Overall responsibility
- Technical Lead: Technical investigation
- Communications Lead: Internal/external communications
- Legal/Compliance: Regulatory requirements
- Support: Additional resources as needed

Response Timeline:

- 0-15 minutes: Acknowledge, assemble IRT, initial assessment
- 15-60 minutes: Containment, evidence preservation
- 1-4 hours: Investigation, communication plan
- 4-24 hours: Eradication, recovery planning

- 24-72 hours: Recovery, monitoring
- 1 week: Post-incident review, improvements

Forensic Requirements:

- Log Preservation: All logs preserved for investigation
- Chain of Custody: Document evidence handling
- Imaging: Disk images of affected systems
- Timeline Analysis: Event reconstruction

Communication Plan:

- Internal: Immediate notification to IRT, executives
- Regulatory: Legal requirements (GDPR 72 hours)
- Customers: Transparent communication based on impact
- Public: Press release if warranted

## 6.8. Compliance & Auditing

Regulatory Compliance:

GDPR Requirements:

- Data Mapping: All personal data identified and documented
- Consent Management: Explicit consent for processing
- Right to Access: Provide data within 30 days
- Right to Erasure: Delete all user data upon request
- Data Protection Officer: Designated responsibility
- Data Processing Agreements: With all processors

PCI DSS Compliance:

- Scope: Cardholder data environment defined
- Requirements: 12 PCI DSS requirements implemented
- Validation: Annual assessment by QSA
- Segmentation: Card data isolated from other systems

Industry Standards:

- ISO 27001: Information security management

- SOC 2: Security, availability, confidentiality
- NIST CSF: Cybersecurity framework
- OWASP: Web application security standards

Auditing & Reporting:

Internal Audits:

- Frequency: Quarterly
- Scope: Random sampling of controls
- Reporting: Findings to security committee
- Remediation: Tracked to completion

External Audits:

- Frequency: Annual
- Auditors: Independent third-party
- Certifications: Maintain relevant certifications
- Transparency: Audit reports available to enterprise customers

Continuous Compliance Monitoring:

- Automated Checks: Daily compliance validation
- Configuration Drift: Alert on non-compliant changes
- Evidence Collection: Automated for audit trails
- Reporting: Real-time compliance dashboard

Documentation Requirements:

- Security Policies: Documented and accessible
- Procedures: Step-by-step for security operations
- Records: Logs, audit trails, incident reports
- Training Materials: Security awareness content

## 6.9. Security Testing

Testing Methodology:

Static Application Security Testing (SAST):

- Tools: SonarQube, ESLint security plugins
- Frequency: On every commit
- Scope: All application code
- Integration: Block PR on critical findings

Dynamic Application Security Testing (DAST):

- Tools: OWASP ZAP, Burp Suite
- Frequency: Weekly automated, monthly manual
- Scope: All exposed endpoints
- Credentials: Test with different user roles

Interactive Application Security Testing (IAST):

- Tools: Contrast Security, Synopsys
- Frequency: During development and testing
- Scope: Runtime application analysis
- Benefits: Real-time vulnerability detection

Penetration Testing:

- Internal: Quarterly by security team
- External: Biannual by third-party
- Scope: Black box, gray box, white box approaches
- Reporting: Detailed findings with remediation guidance

Bug Bounty Program:

- Platform: HackerOne or Bugcrowd
- Scope: In-scope and out-of-scope assets defined
- Rewards: Based on severity and impact
- Process: Clear reporting and response procedures

Security Test Cases:

Authentication Tests:

- Brute force protection
- Session management
- Password policy enforcement
- MFA bypass attempts

Authorization Tests:

- Horizontal privilege escalation
- Vertical privilege escalation
- IDOR (Insecure Direct Object Reference)
- Function-level access control

Input Validation Tests:

- SQL injection
- Cross-site scripting (XSS)
- Command injection
- File upload vulnerabilities

Business Logic Tests:

- Race conditions
- Workflow bypasses
- Payment manipulation
- License validation bypass

Reporting Requirements:

- Vulnerability Reports: Severity, impact, reproduction steps
- Remediation Tracking: From discovery to fix
- Metrics: Time to detect, time to remediate
- Trend Analysis: Vulnerability patterns over time

## 6.10. Security Awareness & Training

Training Program:

Onboarding Training:

- All Employees: Security policies, acceptable use
- Developers: Secure coding practices
- Operations: Infrastructure security
- Support: Customer data handling

Ongoing Training:

- Monthly: Security newsletter with updates
- Quarterly: Phishing simulation exercises
- Biannual: Security workshop or webinar
- Annual: Full security training refresh

Role-Specific Training:

Development Team:

- OWASP Top 10 prevention
- Secure code review practices
- Dependency vulnerability management
- Security testing integration

Operations Team:

- Infrastructure hardening
- Incident response procedures
- Log analysis for security events
- Disaster recovery execution

Management Team:

- Risk management
- Compliance requirements
- Incident communication
- Budgeting for security

Metrics & Evaluation:

- Training Completion: 100% of employees annually
- Phishing Test Results: < 10% click rate target
- Security Quiz Scores: > 80% passing rate
- Incident Response Drills: Quarterly participation

Security Culture:

- Reporting Encouraged: Safe reporting of security concerns
- Recognition: Rewards for security improvements
- Transparency: Open discussion of security incidents

- Continuous Improvement: Feedback loop for security program

## 6.11. Third-party Security

Vendor Assessment Process:

Pre-engagement Assessment:

- Security Questionnaire: Detailed security practices
- Certifications: SOC 2, ISO 27001, etc.
- Penetration Tests: Recent third-party tests
- References: Other customers' experiences

Contractual Requirements:

- Security Clauses: Specific security obligations
- Audit Rights: Right to audit third-party security
- Liability: Clear liability for security breaches
- Termination: Security-based termination rights

Ongoing Monitoring:

- Security Updates: Notification of security incidents
- Compliance Verification: Regular compliance checks
- Performance Monitoring: Service availability and security
- Annual Review: Re-assessment of security posture

Critical Third-party Services:

Payment Processing (Stripe):

- Data Handling: No card data stored locally
- Compliance: PCI DSS Level 1 certification
- Monitoring: Transaction anomaly detection
- Backup: Alternative payment provider identified

AI Generation (Suno):

- Data Protection: Agreement on data usage

- Security: API key protection, rate limiting
- Content Rights: Clear ownership of generated content
- Contingency: Alternative AI providers identified

Hosting & Infrastructure (AWS):

- Shared Responsibility: Clear understanding of responsibilities
- Security Features: Utilization of AWS security services
- Compliance: AWS compliance certifications
- Disaster Recovery: Multi-region deployment

Integration Security:

- API Security: Mutual TLS, API keys, rate limiting
- Data Minimization: Only necessary data shared
- Encryption: Data encrypted in transit and at rest
- Monitoring: Integration point monitoring

## 6.12. Security Metrics & Reporting

Key Security Metrics:

Preventive Metrics:

- Vulnerability Density: Vulnerabilities per 1000 lines of code
- Patch Compliance: % of systems patched within SLA
- Security Training: % of employees trained
- Security Testing Coverage: % of code covered by security tests

Detective Metrics:

- Mean Time to Detect (MTTD): Time from compromise to detection
- Alert Volume: Number of security alerts per day
- False Positive Rate: % of alerts that are false positives
- Monitoring Coverage: % of systems with security monitoring

Responsive Metrics:

- Mean Time to Respond (MTTR): Time from detection to response
- Incident Resolution Rate: % of incidents resolved within SLA
- Containment Time: Time to contain security incidents
- Recovery Time: Time to fully recover from incidents

Predictive Metrics:

- Risk Score: Overall security risk score
- Threat Intelligence: Number of relevant threats detected
- Compliance Score: % of compliance requirements met
- Security Debt: Backlog of security improvements needed

Reporting Schedule:

Daily:

- Security alert summary
- Blocked attack attempts
- System health status

Weekly:

- Vulnerability status report
- Incident response metrics
- Security operations summary

Monthly:

- Security KPI dashboard
- Risk assessment update
- Compliance status report

Quarterly:

- Security program effectiveness
- Third-party security review
- Penetration test results

Annual:

- Security maturity assessment
- Audit results and certifications
- Security roadmap for next year

Executive Reporting:

- Board Level: Strategic risks, program effectiveness
- Management Level: Operational metrics, incident trends
- Technical Level: Detailed findings, remediation status
- Regulatory Level: Compliance evidence, audit reports

# 7: AI-ASSISTED DEVELOPMENT PROTOCOL

```
markdown
## 7. AI-ASSISTED DEVELOPMENT PROTOCOL

### 7.1. AI Development Philosophy

**Core Principles:**
1. **AI as Senior Developer:** Treat AI as experienced team member with specific roles
2. **Human Oversight:** Critical decisions and architecture reviewed by human
3. **Incremental Development:** Build in small, testable increments
4. **Context Preservation:** Maintain development context across sessions
5. **Quality First:** No compromise on code quality or security

**AI Role Definitions:**
- **Architect:** System design, database schema, API design
- **Frontend Developer:** React/Next.js components, state management
- **Backend Developer:** API endpoints, business logic, database queries
```

- **DevOps Engineer:** Infrastructure, deployment, monitoring
- **QA Engineer:** Test creation, bug detection, quality assurance

**Success Metrics:**
- **Code Quality:** > 80% test coverage, < 3% code duplication
- **Development Speed:** 2-3x faster than traditional development
- **Bug Rate:** < 0.1% production defects from AI-generated code
- **Maintainability:** Clear documentation, modular architecture

---

### 7.2. Context Management System

**Context Documentation Structure:**

**Project-Level Context (`/context/project/`):**
- `vision.md`: Business goals, target users, value proposition
- `architecture.md`: System architecture, technology choices
- `standards.md`: Coding standards, naming conventions, patterns
- `glossary.md`: Domain-specific terms and definitions

**Module-Level Context (`/context/modules/{module-name}/`):**
- `purpose.md`: Module responsibility, business value
- `dependencies.md`: Other modules this depends on
- `api.md`: Public API surface, interfaces
- `implementation.md`: Key algorithms, data structures
- `testing.md`: Test strategy, mock data, edge cases

**Task-Level Context (`/context/tasks/{task-id}/`):**
- `requirements.md`: What needs to be built
- `constraints.md`: Limitations, performance requirements
- `acceptance.md`: How to verify completion
- `notes.md`: Progress, decisions, challenges

**Context Update Protocol:**
1. **Before Task:** Read relevant context files
2. **During Development:** Update context with decisions made
3. **After Completion:** Update with learnings, improvements
4. **Weekly Review:** Clean up outdated context

**Context File Template:**
```markdown
# Module: {module-name}

## Purpose
[What this module does, business value]

## Dependencies
- Depends on: [list modules]
- Used by: [list modules]

## API Surface
```typescript
// Interface definitions
```

# Key Implementation Details

- Algorithm: [description]
- Data Structures: [description]
- Performance Considerations: [notes]

# Testing Strategy

- Unit Tests: [coverage areas]
- Integration Tests: [scenarios]
- Edge Cases: [specific cases to test]

# Change History

- YYYY-MM-DD: [change description]
```

```
text
---

### 7.3. Prompt Engineering Framework

**Prompt Template Library:**

**Architecture Prompt Template:**
```

ROLE: Senior Software Architect
TASK: Design {component-name} component/system
CONTEXT: [paste relevant context from /context/]
REQUIREMENTS:

- Functional: [list functional requirements]
- Non-functional: [performance, security, scalability]
- Constraints: [technical/business constraints]
- Integration: [how it integrates with existing system]

DELIVERABLES:

1. Architecture diagram (Mermaid/ASCII)
2. Component responsibilities
3. API design (OpenAPI/TypeScript)
4. Data model (SQL/NoSQL)
5. Security considerations
6. Scaling strategy

CODING STANDARDS:

- Language: TypeScript/Node.js/React
- Framework: Next.js 14 (App Router)
- Styling: Tailwind CSS
- State Management: Zustand
- Testing: Jest + React Testing Library
- Documentation: JSDoc + README

```
text
**Implementation Prompt Template:**
```

ROLE: Senior {language} Developer
TASK: Implement {feature-name} in {module-name}
CONTEXT: [paste architecture and module context]

IMPLEMENTATION DETAILS:

- File: {path/to/file}
- Function: {function-name}
- Input: {parameters}
- Output: {return value}
- Error Handling: {specific error cases}
- Edge Cases: [list edge cases to handle]

CODE REQUIREMENTS:

- Follow existing patterns in codebase
- Include comprehensive error handling
- Add JSDoc comments for public APIs
- Write unit tests for critical paths
- Ensure TypeScript strict mode compliance
- Add to existing test suite

TESTING REQUIREMENTS:

- Unit Tests: Cover all public methods
- Integration Tests: Test with dependencies
- Edge Cases: Handle [specific edge cases]
- Performance: Meet [performance targets]

SECURITY REQUIREMENTS:

- Input validation: [validation rules]
- Authentication: [auth requirements]
- Authorization: [permission checks]
- Data protection: [encryption requirements]

```
text
**Code Review Prompt Template:**
```

ROLE: Senior Code Reviewer
TASK: Review code changes in {file-path}
CONTEXT: [link to PR/commit]

REVIEW CRITERIA:

1. Functionality: Does it meet requirements?
2. Code Quality: Readability, maintainability
3. Performance: Any bottlenecks or inefficiencies?
4. Security: Vulnerabilities, data protection
5. Testing: Adequate test coverage
6. Documentation: Clear comments and docs

SPECIFIC CONCERNS TO CHECK:

- [Specific security concern from requirements]
- [Performance consideration]
- [Integration with existing code]
- [Error handling completeness]

REVIEW OUTPUT FORMAT:

- Overall Assessment: [PASS/FAIL with explanation]
- Critical Issues: [list with line numbers]
- Suggestions: [improvement recommendations]
- Questions: [clarifications needed]

```
text
**Debugging Prompt Template:**
```

ROLE: Senior Debugging Engineer
PROBLEM: {describe the problem}
SYMPTOMS: [observed behavior]
EXPECTED: [expected behavior]
CONTEXT: [system state, recent changes]

DEBUGGING STEPS REQUESTED:

1. Analyze error logs: [paste logs]
2. Review relevant code: [file paths]
3. Check data flow: [input → processing → output]
4. Identify root cause: [hypothesis]
5. Suggest fix: [code changes needed]
6. Prevent recurrence: [long-term solution]

INVESTIGATION AREAS:

- Recent deployments/changes
- Data inconsistencies
- Race conditions
- Resource limitations
- Third-party service issues

DELIVERABLES:

1. Root cause analysis
2. Immediate fix
3. Regression tests
4. Monitoring improvements

```
text
---

### 7.4. Development Workflow

**Task Breakdown Process:**

**Phase 1: Requirement Analysis (Human + AI)**
1. Human provides high-level requirement
2. AI breaks down into technical tasks
3. Human reviews and prioritizes tasks
4. AI estimates effort for each task

**Phase 2: Design & Architecture (AI-led)**
1. AI creates technical design document
2. Human reviews and provides feedback
3. AI updates design based on feedback
4. Final design approved by human

**Phase 3: Implementation (AI-led)**
```

```
1. AI implements smallest testable unit
2. AI writes tests for the implementation
3. AI runs tests and fixes issues
4. Human reviews completed unit

**Phase 4: Integration (AI + Human)**
1. AI integrates unit with existing system
2. AI runs integration tests
3. Human performs smoke testing
4. AI addresses integration issues

**Phase 5: Review & Deployment (Human-led)**
1. Human performs final code review
2. AI generates deployment artifacts
3. Human approves deployment
4. AI monitors post-deployment

**Daily Development Cycle:**
```

Morning (Planning):

1. Review previous day's progress
2. Update context files
3. Plan today's tasks
4. Generate implementation prompts

Afternoon (Development):

1. Execute implementation prompts
2. Run tests and fix issues
3. Update context with progress
4. Prepare for next day

Evening (Review):

1. Review AI-generated code
2. Update acceptance criteria
3. Document learnings
4. Plan improvements

```
text
```

---

### 7.5. Quality Assurance Protocol

**AI Testing Responsibilities:**

**Test Generation Rules:**
- **Unit Tests:** Generated for all public functions/methods
- **Integration Tests:** Generated for critical user flows
- **Edge Cases:** Tests for boundary conditions and error states
- **Performance Tests:** For operations with specific SLAs
- **Security Tests:** For authentication, authorization, validation

**Test Coverage Requirements:**
- **Core Business Logic:** 95%+ coverage
- **API Endpoints:** 90%+ coverage
- **UI Components:** 80%+ coverage
- **Security Functions:** 100% coverage

**Test Data Management:**
- **Fixture Generation:** AI creates realistic test data
- **Data Privacy:** No real user data in tests
- **Data Variety:** Cover different scenarios and edge cases
- **Data Reset:** Tests clean up after themselves

**Automated Quality Gates:**

**Pre-commit Checks:**
- Code formatting (Prettier)
- Linting (ESLint with security rules)
- Type checking (TypeScript)
- Unit tests (affected code only)

**Pre-PR Checks:**
- Full test suite
- Code coverage minimums
- Security vulnerability scan
- Performance benchmark comparison

**Pre-deployment Checks:**
- Integration tests
- Load tests (critical paths)
- Security penetration tests (automated)
- Accessibility compliance checks

**Quality Metrics Tracking:**
- **Defect Density:** Bugs per 1000 lines of code
- **Test Effectiveness:** Bugs caught by tests vs production
- **Code Review Feedback:** AI vs human review consistency
- **Technical Debt:** Identified and tracked

---

### 7.6. Code Review & Validation

**AI Self-Review Process:**

**Before Human Review:**
1. **Static Analysis:** Run linters, security scanners
2. **Test Execution:** Run all relevant tests
3. **Performance Check:** Benchmark critical operations
4. **Security Audit:** Check for common vulnerabilities
5. **Documentation Review:** Ensure completeness

**Review Checklist:**
- [ ] Code follows project conventions
- [ ] No security vulnerabilities introduced
- [ ] Performance requirements met
- [ ] Error handling comprehensive
- [ ] Tests cover edge cases
- [ ] Documentation complete and accurate
- [ ] Backward compatibility maintained
- [ ] No unintended side effects

**Human Review Process:**

**Review Focus Areas:**
1. **Business Logic:** Correctness of implementation
2. **Architecture:** Alignment with system design
3. **Security:** Manual security review
4. **User Experience:** For UI components

```
5. **Maintainability:** Long-term code health

**Review Feedback Format:**
```

# Summary

[Overall assessment]

# Critical Issues (Must Fix)

- Issue 1 with suggested fix
- Issue 2 with suggested fix

# Suggestions (Should Fix)

- Suggestion 1
- Suggestion 2

# Questions

- Question 1
- Question 2

# Approval

[ ] Approved
[ ] Needs Revision

```
text
**Automated Review Tools Integration:**
- **SonarQube:** Code quality metrics
- **Snyk:** Dependency vulnerability scanning
- **CodeQL:** Security analysis
- **Lighthouse:** Web performance and accessibility

---

### 7.7. Knowledge Management
```

**Learning Loop System:**

**Success Pattern Database:**
- **Location:** `/patterns/success/`
- **Content:** Successful implementations with context
- **Format:** Problem → Solution → Result
- **Usage:** Reference for similar future tasks

**Failure Pattern Database:**
- **Location:** `/patterns/failure/`
- **Content:** Failed approaches and lessons learned
- **Format:** Problem → Attempted Solution → Failure Reason → Lesson
- **Usage:** Avoid repeating mistakes

**Decision Log:**
- **Location:** `/decisions/`
- **Content:** Architecture and implementation decisions
- **Format:** Date → Decision → Alternatives Considered → Rationale
- **Usage:** Understand why choices were made

**Continuous Improvement Process:**

**Weekly Retrospective:**
1. Review completed work
2. Identify successful patterns
3. Document failures and lessons
4. Update context files
5. Plan improvements for next week

**Monthly Optimization:**
1. Analyze development metrics
2. Identify bottlenecks
3. Update prompt templates
4. Refine development workflow
5. Train on new patterns

**Quarterly Review:**
1. Assess overall development efficiency
2. Update AI development protocol
3. Incorporate new tools/techniques

4. Set goals for next quarter

**Knowledge Sharing:**
- **Cross-module Learning:** Apply patterns from one area to another
- **Pattern Recognition:** Identify recurring problems and solutions
- **Best Practice Evolution:** Continuously improve development standards
- **Tool Optimization:** Better use of development tools

---

### 7.8. Tooling & Automation

**Development Environment Setup:**

**Required Tools:**
- **IDE:** VS Code with specified extensions
- **Version Control:** Git with conventional commits
- **Package Manager:** pnpm (for monorepo)
- **Containerization:** Docker for local development
- **Database:** Local PostgreSQL + Meilisearch

**VS Code Extensions:**
- TypeScript/JavaScript support
- Tailwind CSS IntelliSense
- ESLint
- Prettier
- GitLens
- Thunder Client (API testing)
- Mermaid preview

**Development Scripts:**

**Local Development:**
```bash
# Start all services
npm run dev:all

# Start specific app
npm run dev:creator-store
```

```
# Run tests
npm run test:watch

# Lint code
npm run lint:fix

# Type check
npm run type-check
```

Code Generation:

```
bash
# Generate new component
npm run generate:component --name=Button --type=ui

# Generate API endpoint
npm run generate:api --name=search --method=get

# Generate test suite
npm run generate:test --file=src/components/Button.tsx
```

AI Development Assistant Tools:

Context Management:

- Tool: Custom CLI for context updates
- Commands:
  - `context update <module> <file>`: Update context
  - `context review`: Review all context files
  - `context sync`: Sync with codebase changes

Prompt Management:

- Tool: Prompt library with templates
- Features:
  - Template storage and retrieval
  - Context variable injection

    - Prompt versioning
    - Effectiveness tracking

Code Quality Automation:

- Pre-commit Hooks: Automated quality checks
- CI/CD Integration: Automated testing and deployment
- Monitoring: Code quality metrics dashboard
- Alerting: Quality degradation notifications

Performance Monitoring:

- Development: Local performance profiling
- Staging: Load testing automation
- Production: Real user monitoring
- Feedback Loop: Performance issues → Test cases

## 7.9. Risk Management

AI Development Risks:

Technical Risks:

1. Code Quality Degradation: AI producing low-quality code
    - Mitigation: Strict review process, automated quality gates
2. Architecture Drift: Inconsistent architecture decisions
    - Mitigation: Architecture review for all major changes
3. Security Vulnerabilities: Introduction of security flaws
    - Mitigation: Security-focused code review, automated scanning

Process Risks:

1. Context Loss: Forgetting previous decisions
    - Mitigation: Comprehensive context management
2. Scope Creep: Uncontrolled feature expansion

    - Mitigation: Clear requirements, change control process
3. Knowledge Siloing: Only AI understands certain code
    - Mitigation: Human review, documentation requirements

Operational Risks:

1. AI Service Outage: Development blocked
    - Mitigation: Local model fallback, cached responses
2. Cost Overruns: Excessive AI API usage
    - Mitigation: Usage tracking, budget alerts
3. Dependency Risk: Over-reliance on specific AI model
    - Mitigation: Multi-model support, abstraction layer

Risk Monitoring:

- Daily: Review AI-generated code quality
- Weekly: Assess development velocity and quality
- Monthly: Review security and architecture compliance
- Quarterly: Full risk assessment and mitigation review

Contingency Plans:

AI Performance Degradation:

1. Switch to alternative AI model/service
2. Increase human review and intervention
3. Fall back to traditional development for critical components
4. Implement additional automated testing

Security Incident from AI Code:

1. Immediate rollback of affected code
2. Security audit of all recent AI-generated code
3. Update AI training to avoid similar issues
4. Implement additional security validation steps

Development Timeline Slippage:

1. Re-prioritize features based on business value

2. Increase human development resources
3. Simplify implementation approaches
4. Adjust scope based on velocity

## 7.10. Success Measurement

Development Metrics:

Efficiency Metrics:

- Lines of Code per Hour: AI vs human comparison
- Task Completion Rate: % of tasks completed on time
- Rework Percentage: % of code requiring significant changes
- Development Velocity: Features delivered per week

Quality Metrics:

- Defect Rate: Bugs per 1000 lines of code
- Test Coverage: % of code covered by tests
- Code Review Pass Rate: % of code approved on first review
- Technical Debt Index: Quantified technical debt

Economic Metrics:

- Development Cost: Cost per feature/component
- ROI: Value delivered vs development cost
- Maintenance Cost: Ongoing cost of AI-generated code
- Total Cost of Ownership: Development + maintenance cost

Satisfaction Metrics:

- Developer Experience: Ease of working with AI-generated code
- Code Understandability: Human comprehension of AI code
- System Reliability: Uptime and error rates
- Business Value Delivered: Features that meet user needs

Continuous Improvement Framework:

Feedback Collection:

- Daily: Quick feedback on AI assistance
- Weekly: Development experience survey
- Monthly: Code quality and productivity review
- Quarterly: Comprehensive process evaluation

Improvement Implementation:

1. Identify: Areas for improvement from metrics and feedback
2. Prioritize: Based on impact and effort
3. Implement: Changes to process, tools, or prompts
4. Measure: Impact of improvements
5. Iterate: Continuous refinement

Benchmarking:

- Internal: Compare different AI approaches/methods
- External: Compare with industry benchmarks
- Historical: Track improvement over time
- Goal-based: Progress toward specific targets

Reporting Structure:

- Daily Status: What was accomplished, any issues
- Weekly Summary: Metrics, learnings, improvements
- Monthly Review: Comprehensive analysis and planning
- Quarterly Business Review: Strategic assessment and direction

# 8: PHASED ROLLOUT & VALIDATION MATRIX

```
markdown
## 8. PHASED ROLLOUT & VALIDATION MATRIX

### 8.1. Rollout Philosophy & Principles

**Core Rollout Principles:**
1. **Risk-Managed Progression:** Each phase must succeed
before next begins
```

2. **Feedback-Driven Evolution:** User feedback shapes subsequent phases
3. **Quality Over Speed:** No phase advancement with critical issues
4. **Business Value Focus:** Each phase delivers tangible value
5. **Scalability Validation:** Each phase tests next scale level

**Success Criteria Framework:**
- **Technical Success:** System stability, performance, security
- **Business Success:** User adoption, engagement, revenue
- **Operational Success:** Support capacity, monitoring, incident response
- **Strategic Success:** Market validation, competitive positioning

**Phase Gate Requirements:**
- All critical bugs resolved (P0, P1)
- Performance SLAs met for phase scale
- Security audit passed
- User satisfaction threshold achieved
- Business metrics on track

---

### 8.2. Phase 1: Foundation & MVP (Weeks 1-6)

**Objective:** Validate core value proposition with minimal feature set

**Scope Boundaries:**
- **Included:**
  - Creator Store: Basic search, licensing, download
  - Admin Factory: Track approval, basic QC
  - Core Infrastructure: Database, search, storage
- **Excluded:**
  - Venue Player (B2B)
  - Advanced features (432Hz, analytics, disputes)
  - Mobile optimization
  - Performance optimizations

**Technical Milestones:**

**Week 1–2: Foundation**
- [ ] Monorepo setup with basic structure
- [ ] Database schema implemented
- [ ] Authentication system working
- [ ] Development environment complete

**Week 3–4: Core Features**
- [ ] Basic search with Meilisearch
- [ ] Track licensing workflow
- [ ] Admin QC interface
- [ ] Payment integration (test mode)

**Week 5–6: Polish & Launch Prep**
- [ ] End-to-end testing
- [ ] Performance optimization
- [ ] Security hardening
- [ ] Production deployment

**Validation Criteria:**

**Technical Validation:**
- **Performance:** Search < 200ms, Licensing < 2s
- **Reliability:** 99.5% uptime, error rate < 0.5%
- **Security:** No critical vulnerabilities, secure auth
- **Scalability:** Support 100 concurrent users

**Business Validation:**
- **User Acquisition:** 50 creators onboarded
- **Engagement:** 20% license conversion rate
- **Quality:** 100 tracks in catalog, 80% approval rate
- **Revenue:** First $100 in revenue

**Operational Validation:**
- **Monitoring:** Core metrics dashboard active
- **Support:** Basic support process established
- **Deployment:** CI/CD pipeline working
- **Backup:** Daily backups verified

**Phase Exit Criteria:**
- [ ] 30 days of stable operation
- [ ] No critical (P0) bugs for 7 days

- [ ] User satisfaction score > 4/5
- [ ] Business metrics trending positively

---

### 8.3. Phase 2: B2B Expansion (Weeks 7–10)

**Objective:** Launch venue player and validate B2B model

**New Features:**
- **Venue Player:** Web Audio player, schedule management
- **Offline Mode:** Cache management, background sync
- **Schedule Manager:** Time-based programming
- **Basic Analytics:** Play counts, skip rates

**Technical Enhancements:**
- PWA capabilities for venue app
- Web Audio API integration
- IndexedDB for offline storage
- Real-time sync mechanisms

**Scale Targets:**
- **Users:** 10 venues, 200 creators
- **Concurrency:** 50 venue players, 500 creator sessions
- **Data:** 500 tracks, 1000 licenses
- **Performance:** Audio streaming < 100ms latency

**Validation Criteria:**

**Technical Validation:**
- **Offline Reliability:** 24+ hours continuous playback
- **Audio Quality:** Gapless playback, no audio artifacts
- **Sync Performance:** Playlist sync < 30 seconds
- **Memory Management:** No memory leaks in 24h playback

**Business Validation:**
- **Venue Adoption:** 5 venues using daily
- **Usage Metrics:** > 4 hours/day average playback
- **Retention:** 80% venue retention after 30 days
- **Upsell:** 30% of venues interested in premium features

**Operational Validation:**
- **Monitoring:** Audio-specific metrics (buffering, skips)

- **Support:** Venue-specific support materials
- **Deployment:** Zero-downtime updates verified
- **Capacity:** Handled planned scale with 50% headroom

**Risk Mitigation:**
- **Audio Issues:** Fallback to HTML5 Audio
- **Sync Failures:** Manual sync option
- **Memory Problems:** Automatic session refresh
- **Scale Issues:** Gradual venue onboarding

**Phase Exit Criteria:**
- [ ] 10 venues using system for 14+ days
- [ ] No audio-related critical bugs for 14 days
- [ ] Venue satisfaction score > 4/5
- [ ] System handles 2x current load in load tests

---

### 8.4. Phase 3: Feature Enhancement (Weeks 11-14)

**Objective:** Add advanced features and optimize user experience

**New Features:**
- **432Hz Engine:** Real-time frequency transposition
- **Advanced Search:** Visual query builder, saved searches
- **Analytics Dashboard:** Business intelligence for venues
- **Dispute Center:** YouTube Content ID support
- **Mobile Optimization:** Responsive design improvements

**Technical Enhancements:**
- Real-time audio processing (432Hz)
- Advanced search algorithms
- Analytics pipeline and dashboards
- Mobile-first responsive design

**Scale Targets:**
- **Users:** 50 venues, 1000 creators
- **Concurrency:** 200 venue players, 2000 creator sessions
- **Data:** 2000 tracks, 5000 licenses
- **Performance:** Complex searches < 300ms, 432Hz conversion < 500ms

**Validation Criteria:**

**Technical Validation:**
- **Audio Processing:** 432Hz conversion quality > 95% accuracy
- **Search Relevance:** Search result relevance score > 90%
- **Analytics Performance:** Dashboard loads < 2s with real data
- **Mobile Experience:** Core mobile flows < 3s load time

**Business Validation:**
- **Feature Adoption:** 40% using advanced features
- **Engagement Increase:** 25% increase in search/licensing
- **Revenue Growth:** 50% month-over-month growth
- **Customer Satisfaction:** NPS score > 30

**Operational Validation:**
- **Monitoring:** Feature-specific usage metrics
- **Support:** Advanced feature documentation
- **Performance:** System handles feature complexity
- **Quality:** Automated testing for new features

**Specialized Testing:**
- **Audio Quality:** ABX testing for 432Hz vs 440Hz
- **Search Relevance:** Human evaluation of search results
- **Mobile Usability:** User testing on target devices
- **Analytics Accuracy:** Data validation against source systems

**Phase Exit Criteria:**
- [ ] All advanced features stable for 14 days
- [ ] User adoption of new features > 30%
- [ ] Technical metrics meet or exceed targets
- [ ] No regression in core functionality

---

### 8.5. Phase 4: Scale & Monetization (Weeks 15–18)

**Objective:** Scale system, optimize monetization, prepare for growth

**Focus Areas:**

- **Performance Optimization:** Caching, CDN, database tuning
- **Monetization Features:** Subscription tiers, upsell paths
- **Scale Infrastructure:** Auto-scaling, multi-region readiness
- **Enterprise Features:** API access, white-label options

**Technical Enhancements:**
- Advanced caching strategies
- Multi-region database replication
- API rate limiting and management
- Subscription billing system

**Scale Targets:**
- **Users:** 200 venues, 5000 creators
- **Concurrency:** 1000 venue players, 5000 creator sessions
- **Data:** 10,000 tracks, 25,000 licenses
- **Performance:** P99 latency < 500ms under load

**Validation Criteria:**

**Technical Validation:**
- **Scale Performance:** Handles 10x current load in stress tests
- **Cost Efficiency:** Infrastructure cost per user decreasing
- **Reliability:** 99.9% uptime over 30 days
- **Security:** Penetration test with no critical findings

**Business Validation:**
- **Monetization:** 20% conversion to paid tiers
- **Growth Rate:** Sustained 20% week-over-week growth
- **Unit Economics:** CAC < LTV/3
- **Market Position:** Clear competitive differentiation

**Operational Validation:**
- **Automation:** 90% of operational tasks automated
- **Monitoring:** Predictive alerting working
- **Support:** Scalable support processes
- **Disaster Recovery:** DR tests completed successfully

**Growth Readiness:**
- **Infrastructure:** Can scale 10x with minimal intervention
- **Processes:** Support, onboarding, billing scaled
- **Team:** Knowledge transfer completed
- **Documentation:** Comprehensive for all systems

**Phase Exit Criteria:**
- [ ] System scales to 10x current load
- [ ] Unit economics positive and improving
- [ ] Operational processes scaled and documented
- [ ] Growth runway for 6+ months established

---

### 8.6. Phase 5: Ecosystem & Platform (Beyond Week 18)

**Objective:** Expand beyond core product to ecosystem

**Strategic Initiatives:**
- **API Platform:** Third-party developer access
- **Marketplace:** User-generated content marketplace
- **Integration Ecosystem:** Connect with other platforms
- **Hardware:** AuraBox device development
- **International Expansion:** Multi-language, multi-currency

**Technical Initiatives:**
- Public API with documentation
- Webhook system for integrations
- Multi-tenant architecture
- Hardware SDK and APIs

**Business Objectives:**
- **Platform Revenue:** 30% from ecosystem/API
- **Network Effects:** User-generated content growth
- **Strategic Partnerships:** 5+ major integrations
- **Market Expansion:** 3+ new geographic markets

**Validation Framework:**
- **Developer Adoption:** 100+ API developers
- **Ecosystem Health:** Growing third-party integrations
- **Platform Stability:** API uptime > 99.95%

- **Strategic Value:** Clear path to market leadership

---

### 8.7. Validation Matrix & Success Metrics

**Comprehensive Validation Framework:**

**Technical Metrics Matrix:**
| Metric | Phase 1 Target | Phase 2 Target | Phase 3 Target | Phase 4 Target |
|---|---|---|---|---|
| API Response (P95) | < 500ms | < 300ms | < 200ms | < 100ms |
| Uptime | 99.5% | 99.7% | 99.8% | 99.9% |
| Error Rate | < 0.5% | < 0.3% | < 0.2% | < 0.1% |
| Concurrent Users | 100 | 500 | 2000 | 5000 |
| Data Volume | 1GB | 10GB | 50GB | 200GB |

**Business Metrics Matrix:**
| Metric | Phase 1 Target | Phase 2 Target | Phase 3 Target | Phase 4 Target |
|---|---|---|---|---|
| MAU | 100 | 500 | 2000 | 10000 |
| Revenue/Month | $500 | $2000 | $10000 | $50000 |
| Conversion Rate | 5% | 10% | 15% | 20% |
| Churn Rate | < 10% | < 8% | < 6% | < 4% |
| NPS Score | > 20 | > 30 | > 40 | > 50 |

**Operational Metrics Matrix:**
| Metric | Phase 1 Target | Phase 2 Target | Phase 3 Target | Phase 4 Target |
|---|---|---|---|---|
| MTTR | < 4 hours | < 2 hours | < 1 hour | < 30 minutes |
| Support Response | < 24 hours | < 12 hours | < 4 hours | < 1 hour |
| Deployment Frequency | Weekly | 3x/week | Daily | Multiple daily |
| Change Failure Rate | < 10% | < 7% | < 5% | < 3% |
| Test Coverage | > 70% | > 75% | > 80% | > 85% |

**Validation Checkpoints:**

**Daily Checkpoints:**
- System health dashboard review
- Error rate and performance trends
- User feedback collection
- Business metrics snapshot

**Weekly Checkpoints:**
- Comprehensive system review
- User satisfaction analysis
- Progress vs phase goals
- Risk assessment update

**Monthly Checkpoints:**
- Phase progress assessment
- Business model validation
- Technical debt review
- Strategic alignment check

**Phase Transition Checkpoints:**
- All phase exit criteria met
- Next phase planning complete
- Team readiness assessment
- Risk mitigation plans updated

---

### 8.8. Risk Management by Phase

**Phase-Specific Risks & Mitigations:**

**Phase 1 Risks:**
- **Technical:** Core architecture flaws
  - Mitigation: Extensive testing, architectural reviews
- **Market:** Value proposition not validated
  - Mitigation: Early user feedback, pivot if needed
- **Operational:** Unable to support early users
  - Mitigation: Limited launch, manual processes

**Phase 2 Risks:**
- **Technical:** Audio streaming reliability

  - Mitigation: Fallback mechanisms, extensive testing
- **Business:** B2B adoption slower than expected
  - Mitigation: Targeted outreach, incentive programs
- **Operational:** Scale of support requirements
  - Mitigation: Documentation, self-service tools

**Phase 3 Risks:**
- **Technical:** Feature complexity causing instability
  - Mitigation: Feature flags, gradual rollout
- **Business:** Advanced features not valued by users
  - Mitigation: User research, iterative development
- **Operational:** Monitoring gap for new features
  - Mitigation: Feature-specific monitoring from day 1

**Phase 4 Risks:**
- **Technical:** Scaling limitations emerge
  - Mitigation: Load testing, architectural improvements
- **Business:** Monetization resistance
  - Mitigation: Value communication, tiered pricing
- **Operational:** Process bottlenecks at scale
  - Mitigation: Automation, process redesign

**Contingency Planning:**
- **Phase Rollback:** Ability to revert to previous phase
- **Feature Disable:** Quickly disable problematic features
- **Scale Back:** Reduce user load if capacity issues
- **Pivot Option:** Alternative business model if needed

---

### 8.9. Team & Resource Planning

**Phase-Based Team Scaling:**

**Phase 1 (Core Team):**
- **Roles:** Lead Developer, Product Manager, Designer
- **Focus:** MVP development, user research
- **Time Allocation:** 100% on Phase 1 deliverables

**Phase 2 (Expanded Team):**
- **New Roles:** Audio Engineer, QA Engineer
- **Focus:** B2B features, reliability
- **Time Allocation:** 70% new features, 30% maintenance

**Phase 3 (Feature Team):**
- **New Roles:** Data Analyst, Mobile Specialist
- **Focus:** Advanced features, optimization
- **Time Allocation:** 50% features, 30% optimization, 20% maintenance

**Phase 4 (Scale Team):**
- **New Roles:** DevOps Engineer, Growth Marketer
- **Focus:** Scaling, monetization, automation
- **Time Allocation:** 30% scale, 30% features, 40% optimization/maintenance

**Resource Requirements by Phase:**

**Infrastructure Costs:**
- **Phase 1:** $500/month (development, basic hosting)
- **Phase 2:** $1500/month (increased scale, audio streaming)
- **Phase 3:** $3000/month (advanced features, analytics)
- **Phase 4:** $5000/month (scale infrastructure, redundancy)

**Tooling Costs:**
- **Phase 1:** $300/month (monitoring, error tracking)
- **Phase 2:** $500/month (additional monitoring, support tools)
- **Phase 3:** $1000/month (analytics, advanced tooling)
- **Phase 4:** $2000/month (enterprise tools, automation)

**External Services:**
- **Phase 1:** Suno API ($500 limit), Stripe (pay-as-you-go)
- **Phase 2:** Increased API limits, additional services
- **Phase 3:** Premium APIs, specialized services
- **Phase 4:** Enterprise agreements, volume discounts

---

### 8.10. Communication & Stakeholder Management

**Stakeholder Communication Plan:**

**Internal Stakeholders:**
- **Daily:** Development team sync, progress update
- **Weekly:** Leadership team review, metric dashboard
- **Monthly:** All-hands update, strategic review
- **Phase Gates:** Formal review and approval

**External Stakeholders:**
- **Early Users:** Weekly check-ins, feedback sessions
- **Investors:** Monthly updates, milestone reports
- **Partners:** Quarterly reviews, collaboration planning
- **Market:** Phase launches, feature announcements

**Progress Reporting:**

**Dashboard Metrics:**
- Real-time system health
- User engagement metrics
- Business performance indicators
- Phase progress tracking

**Report Formats:**
- **Executive Summary:** One-page highlight report
- **Technical Deep Dive:** Detailed technical progress
- **User Feedback:** Consolidated user insights
- **Financial Update:** Budget vs actual, forecasts

**Decision Making Framework:**

**Decision Types:**
- **Tactical:** Day-to-day development decisions (team-level)
- **Strategic:** Phase direction, feature prioritization (leadership)
- **Architectural:** Technology choices, major refactors (tech lead)
- **Business:** Pricing, partnerships, growth (executive)

**Decision Process:**
1. Problem/opportunity identification
2. Options analysis with pros/cons
3. Recommendation based on phase goals
4. Approval by appropriate decision-makers
5. Implementation and follow-up

**Change Management:**
- **Scope Changes:** Formal change request process
- **Timeline Adjustments:** Impact analysis required
- **Resource Reallocation:** Business case justification
- **Phase Reprioritization:** Strategic review and approval

---

### 8.11. Post-Phase Evaluation

**Phase Retrospective Process:**

**Evaluation Framework:**
- **What worked:** Successful patterns, effective approaches
- **What didn't:** Challenges, failures, bottlenecks
- **Learnings:** Key insights about users, technology, market
- **Improvements:** Changes for next phase

**Evaluation Areas:**

**Technical Evaluation:**
- Architecture decisions effectiveness
- Technology choices validation
- Performance against targets
- Technical debt accumulation

**Business Evaluation:**
- Market response validation
- User adoption patterns
- Revenue model effectiveness
- Competitive positioning

**Process Evaluation:**
- Development efficiency
- Team collaboration
- Decision making quality
- Risk management effectiveness

**Improvement Implementation:**
1. **Prioritize:** Based on impact on next phase

2. **Plan:** Specific actions, owners, timelines
3. **Implement:** During phase transition or early next phase
4. **Verify:** Improvement effectiveness measured

**Knowledge Transfer:**
- **Documentation:** Updated based on learnings
- **Training:** Team knowledge sharing sessions
- **Patterns:** Successful approaches codified
- **Anti-patterns:** Failed approaches documented to avoid

**Celebration & Recognition:**
- **Milestone Achievement:** Phase completion celebration
- **Individual Contributions:** Recognition of key efforts
- **Team Success:** Collective achievement acknowledgment
- **Learning Culture:** Valuing both successes and failures as learning opportunities

NOTLAR

**Gelişmiş Arama için önerilerim:**

**Elasticsearch** veya **Meilisearch** entegrasyonu (Supabase'in native search'ü komplex filtreleme için yetersiz kalabilir)
**Fuzzy search** (yazım hataları toleransı)
**Query builder** UI'ı (kullanıcı "BPM: 90-110 AND mood: Chill OR Ambient" gibi mantıksal sorgular oluşturabilsin)
**Saved searches** özelliği (Creator'lar sık kullandıkları filtreleri kaydedebilsin)
**Search analytics** (hangi aramaların sonuç bulamadığı - content gap analizi için)
**Geliştirme sırasında dökümantasyonda eksik bulduğum ve işine yarayabilecek konular:**

Performance monitoring stratejisi
Error tracking & logging
Database backup & disaster recovery planı
CDN cache invalidation stratejisi
API rate limiting rules
Test coverage hedefleri