### **EDM Shuffle: Professional Development Manual**

#### **1. Executive Summary**

**EDM Shuffle is a next-generation cultural-tech platform designed for Gen Z and millennial electronic dance music (EDM) fans, shuffle dancers, DJs, and festival-goers**. Its core mission is to empower creators, dancers, DJs, and fans to express themselves through personalized archetypes, immersive digital tools, and a community-driven festival ecosystem. The platform aims to be the definitive digital destination for EDM lovers, blending creative tools, virtual identity, curated content, and commerce into a gamified, expressive experience.

The project is undergoing a **rebuild from scratch**, prioritizing functionality, AI integration from day one, deep personalization, and archetype-based UI and content, moving past previous iterations with simulated features. EDM Shuffle seeks to unite music, dance, fashion, and event culture into one cohesive system driven by autonomous agents and user-centric personalization.

#### **2. Authoritative Project Roadmap: Development Phases & Milestones**

This roadmap consolidates the latest, unified checklist of deliverables for each development phase, including associated tasks, features, and prompts, based on the most recent instructions.

##### **PHASE 1: Core Infrastructure & Theming (Initial Scaffolding & User Authentication)**

* **Goal:** Establish the foundational UI, visual theme, and core user authentication.
* **Completed Requirements:**
  + Basic React + TypeScript + Tailwind setup with shadcn/ui components.
  + Enhanced Dark Theme with neon color palette (purple/cyan/bass-blue gradients) and custom CSS for glow effects, shimmer, and neon utility classes.
  + Framer Motion installed for smooth transitions, custom keyframes for equalizer, floating elements, glow effects, and scroll-triggered animations.
  + Pages scaffolded: Home, Archetype Quiz, Shuffle Feed, Marketplace Grid, Festival Voting Stage.
  + Bottom navigation bar with glowing icons (Explore, Challenges, Marketplace, Festival).
  + Reusable UI components like NeonButton, GlowCard, EqualizerBars, and FloatingElements (sneakers, particles).
  + Homepage with kinetic festival visuals, PLUR orbs, floating sneakers animation, and a "Find Your Shuffle Archetype" CTA.
  + Archetype Quiz transformed into an immersive cosmic portal experience with multi-step animations, dynamic aura meter, ceremonial reveals, and artist spotlights.
  + Basic user authentication flow with Supabase (sign up, login, email/password reset, user profile).
* **Still Pending/Needs Real Work:**
  + Hero / Homepage: Sound not interactive, equalizer not synced, PLUR orbs don't serve a function beyond visual styling.
  + Archetype Quiz: No connection to feed logic; quiz is still gating, not enhancing overall experience. Archetype selection needs to truly impact site theme/user experience and unlock special content.
  + **Crucially, ensure all audio feedback claimed in prompt updates is actually functional and not simulated (e.g., sound for equalizer, turbtables)**.
* **Validation:** Verify visual fidelity, responsive design, and basic UI interactivity. For authentication, ensure successful user registration, login, and profile display with real (Supabase) data.

##### **PHASE 2: Audio Engine & Mix Station**

* **Goal:** Implement a true Web Audio API engine for the DJ Mix Station with real playback, mixing, effects, and synchronized visualizations.
* **Deliverables:**
  + **Implement true Web Audio API engine:** No placeholders, ensuring robust, scalable audio processing for independent deck control, crossfading, pitch, and echo effects.
  + **Enable Deck A/B real playback, crossfade, pitch, echo FX:** Each deck must play its own track with separate audio elements, gain nodes, delay for echo, playbackRate for pitch, and real-time volume adjustment.
  + **Real-time waveform + BPM visualization synced to audio:** Use Web Audio API AnalyserNode to display real frequency data as animated bars and a master tempo control that affects visual animations.
  + **Crowd Engagement Features:** "Drop My Set" button triggers confetti, light bursts, floating emojis, and toast notifications synchronized with real crowd cheer sound effects, with a mute toggle option.
  + **Debug HUD:** A toggleable development panel showing real-time deck states, audio context status, BPM values, and active settings.
* **Innovation:** Dynamic track loading, multi-browser compatibility, latency optimization, and custom audio effects chains (e.g., reverb, flanger, distortion).
* **Validation:** Thorough cross-browser testing (Chrome, Edge, Safari, Firefox) for audio functionality and visual synchronization. Debug HUD must show real audio engine state.

##### **PHASE 3: Voting & Festival Scheduling**

* **Goal:** Establish real-time festival scheduling, event management, and a robust, rate-limited voting system for DJs/performers, culminating in a headliner selection.
* **Deliverables:**
  + **Real Voting Logic:**
    - Implement rate-limited, unique-per-user voting (e.g., Supabase/Postgres with user ID and timestamp checks).
    - Real-time vote aggregation and display.
    - Anti-spam measures.
  + **Festival Schedule UI + Backend Data Model:**
    - Database schema for Festivals, Events, and Lineups (Supabase).
    - Calendar UI (React component) to display events by date, time, genre, and location.
    - API endpoints for fetching/updating schedule data.
  + **Event Creation/Joining, Headliner Logic, Live Leaderboard:**
    - Users can RSVP/join events.
    - Headliner selection based on vote thresholds with animated crowning.
    - Dynamic leaderboard updating in real-time.
* **Innovation:** Archetype-themed festival "vibe" selection, AI-powered content (e.g., simulated event announcements), localized event recommendations, and NFT ticket integration for future phases.
* **Validation:** Test voting limits, schedule accuracy across timezones, and seamless animations for leaderboards and headliner announcements.

##### **PHASE 4: Challenges & Social**

* **Goal:** Transform the Shuffle Feed into an interactive remix dojo with dynamic challenges, user submissions, and a gamified PLUR/crew system.
* **Deliverables:**
  + **Challenge Feed - Backend/API & Frontend Integration:**
    - Backend for dynamic challenge creation, storage, and retrieval.
    - API endpoints for challenge types, rules, and associated media.
    - Frontend to display daily/weekly challenges, filtered by archetype/difficulty.
  + **Challenge Submission (User Generated Content - UGC):**
    - File upload system for video/audio (Supabase Storage).
    - Submission form for captions, tags (archetype, genre, PLURcrew).
    - Moderation queue for user submissions.
  + **Gamified PLUR System & Leaderboard:**
    - PLURstreak System: Top-left counter for daily flow streak and PLUR meter progression with emoji ladder.
    - Integrate leaderboard for PLUR and challenge winners.
    - PLURpower boost metrics on video tiles.
  + **Remix Mechanics:**
    - "Remix This" button on video tiles that summons a RemixModal with pitch/tempo sliders.
    - Generate button in RemixModal returns a new stub .mp3 URL (mock /api/remix/mock).
    - Update feed item's audio source to remix URL and tag "🔀 Remixed".
* **Innovation:** AI-powered challenge suggestions (e.g., based on trending music), multi-track remixing with in-browser tools, archetype-specific bonus points, and community-driven content moderation.
* **Validation:** Test end-to-end challenge creation, submission, and display. Verify leaderboard updates reflect real progress. Ensure remix mock logic is robust.

##### **PHASE 5: Marketplace**

* **Goal:** Build a functional e-commerce marketplace for ravewear, digital collectibles, and eventually peer-to-peer resale, integrated with payment processing.
* **Deliverables:**
  + **Stripe or other payments integration:**
    - Frontend integration for secure checkout flow.
    - Backend for managing payments, orders, refunds.
  + **Real Product Listings:**
    - Database for products (physical/digital), inventory, pricing, images, descriptions, archetype tags.
    - Admin interface for product management.
    - Frontend displays actual products, not placeholders.
  + **Order Confirmation & Digital Download/Gift Flows:**
    - Order confirmation page/email.
    - Mechanism for digital product delivery (e.g., links, downloads).
    - Support for gifting items to other users.
  + **User Marketplace (Resale):**
    - Users can upload, tag, price, and sell worn outfits, kandi kits, LED accessories, etc., with AI style-matching.
* **Innovation:** AI recommendations for fashion (e.g., style-matching by archetype), virtual try-on features, NFT-backed digital collectibles, and a transparent peer-to-peer resale system for ravewear.
* **Validation:** Test full purchase flow (add to cart, checkout, payment, confirmation). Verify digital/physical item delivery.

##### **PHASE 6: Community & Crews**

* **Goal:** Implement robust social features, including live chat, squad/fam groups, and enhanced user interactions, to foster community.
* **Deliverables:**
  + **Live Chat System:**
    - Scalable, secure real-time chat (Supabase Realtime, Firebase, or Socket.io).
    - Support for global Festival chat, private Squad/Fam chat, and 1:1 DMs.
    - Responsive chat components with auto-scroll, unread badges, and offline error states.
    - Rate-limiting, @mentions, emoji, and markdown support (with input sanitization).
  + **Squad/Fam Features:**
    - User-creatable and joinable "PLURcrew" groups with members displayed as animated cards (power meters, mood bars, avatar slots).
    - Crew management tools (invite, accept, kick, roles).
    - Inter-squad events and challenges.
  + **Avatar & Crew Management:**
    - Avatar customization and evolution mechanics.
    - "Avatar Summon System" for new crew members (20% random chance on interaction, with confetti/modal reveal).
  + **User-to-User Interactions:**
    - Commenting on content, direct messaging.
* **Innovation:** Archetype-specific chat channels, gamified crew battles (e.g., "PLURcrew Showdowns"), AI-moderation for chat, and dynamic "mood bars" for avatars based on content interaction.
* **Validation:** Test real-time chat functionality, group creation/joining, and avatar summoning mechanics.

##### **PHASE 7: Admin + Docs**

* **Goal:** Develop an internal admin dashboard for content and user management, and ensure all project documentation is comprehensive, accurate, and maintained.
* **Deliverables:**
  + **Admin Dashboard:**
    - Role-based access (admin, moderator, content manager).
    - Management for Festival schedule, user accounts (ban, promote, reset passwords), crew/squad approvals, challenges (approve, remove), product listings, comments, and abuse reports.
    - Live activity feed and audit logs for admin actions.
    - Mobile-friendly, secure login (with 2FA option).
  + **Documentation Maintenance:**
    - Keep all project documentation up to date in the root directory.
    - Regular cross-checks of FRS and Masterplan after every milestone.
    - Ensure docs reflect real status, no placeholders without explicit labeling.
* **Innovation:** AI-powered content moderation tools, automated document generation/validation (e.g., FRS from code comments), and real-time dashboard analytics for engagement.
* **Validation:** Test role-based access, CRUD operations for all managed entities, and audit log accuracy. Verify documentation is current and matches implemented features.

##### **PHASE 8: Built-in Marketing Framework**

* **Goal:** Create an in-app marketing suite for campaign management, copy generation, and direct automation handoff to external platforms (n8n, Printify, social media).
* **Deliverables:**
  + **Integrated Marketing Dashboard:**
    - Secure, role-based access from admin menu.
    - Centralized campaign planning, creative copywriting, and social automation.
    - Ability to build, preview, and deploy multi-platform campaigns.
    - Generate, approve, and refine copy with built-in AI tools.
    - Schedule posts, product launches, and merch drops.
    - Direct integration with n8n or other automation for one-click distribution.
  + **Campaign Management:** Define target audience, set goals, track performance.
  + **AI Copywriter:** Generate marketing copy for social media, email, product descriptions, event announcements (leveraging SDXL, ElevenLabs, Stability, Replicate for text/image generation).
  + **Automation Hooks:** Direct connection for n8n to send campaigns to social media (Instagram, TikTok, Pinterest) and Printify for merch drops (e.g., Firestorm Day themed merch).
* **Innovation:** AI-driven campaign optimization, real-time sentiment analysis from social feeds, and automated influencer outreach suggestions.
* **Validation:** Test campaign creation, AI copy generation, and successful deployment via n8n to target platforms.

#### **3. Document Index (Root Directory)**

All official project documentation files are/will be contained in the root directory, ensuring easy access and a single source of truth for all contributors.

* **CLAUDE\_INTEGRITY\_RULES.md:** Core code-of-conduct and workflow constraints for all LLM contributors, ensuring real features and no placeholders. **Status: Done.**
* **IMPLEMENTATION\_MICROTASKS.md:** Exhaustive, sequenced roadmap of every sub-task required for EDM Shuffle, divided by phase. **Status: Done.**
* **PRD.md:** Product Requirements Document. Business goals, audience, feature list, MVP/launch scope. **Status: Done (Rewritten).**
* **FRS.md:** Functional Requirements Spec. Technical feature breakdown, user stories, acceptance criteria. **Status: Done.**
* **MASTERPLAN.md:** High-level architectural masterplan and project vision. **Status: Done (Exists, can be refined based on latest phases).**
* **claudeupdate.md:** Living status ledger and changelog for every microtask/commit, used for cross-LLM accountability. **Status: Done.**
* **AUDIO\_UI\_COMPONENTS.md:** Outline or description of key Audio UI components to be built/verified. **Status: Done (Exists, can be refined).**
* **README.md:** Entry-point for new contributors (setup, rules, tooling, context for LLM+human handoff). **Status: Done.**
* **project\_overview.md:** One-page project intro/summary for quick onboarding. **Status: Done.**
* **technical\_architecture.md:** High-level technical architecture, stack, workflow, key components, LLM/CLI integration points. **Status: Done.**
* **UI\_UX\_STRUCTURE.md:** Component layout, navigation flows, interaction notes. **Status: Done.**
* **CrowdFXLayer.md:** Documentation for crowd effects (confetti, emoji FX, cheer audio). **Status: Done.**
* **SubscribeModal.md:** Documentation for the subscribe CTA and modals. **Status: Done.**
* **DOCUMENTATION\_INDEX.md:** Table of contents for all docs. **Status: Pending (To be generated last, when all other docs are finalized).**
* **FEATURE\_REALITY\_MATRIX.md:** Simple markdown doc showing current vs. real feature status, updated after each sprint/turn. **Status: Pending (Proposed new doc).**
* **USER JOURNEY MAP.md:** Shows actual possible journeys and where they break due to missing functionality. **Status: Pending (Proposed new doc).**
* **MVP\_DEFINITION.md:** Clarifies the minimum viable product boundary. **Status: Pending (Proposed new doc).**

#### **4. Prompt Catalog**

This section collates explicit LLM prompt instructions, prioritizing the final/most complete versions, along with their validation steps and integrity rules.

* **Initial Scaffold & Visuals (Prompt 1, HomePage):**
  + **Prompt:** "Build a modern React + Tailwind + Framer Motion app scaffold... Homepage with kinetic festival visuals, CTA sections, real navigation header... Dark mode default with neon accent theme... NavBar and Hero with animated background (Framer Motion), disco ball or audio visualizer welcome block... Set up routing with modular components per page... Include archetype-switcher dropdown in Nav (placeholder).".
  + **Validation:** Verify visual styling, PLUR orbs, glowing sneakers, basic UI, and responsive design.
* **Archetype Quiz (Prompt 2, ArchetypeQuiz.tsx):**
  + **Prompt:** "Transform the ArchetypeQuiz into an immersive cosmic portal experience with multi-step animations, dynamic aura meters, and a ceremonial reveal... Artist spotlights, matching DJs to the result... Use Tailwind gradients, Framer Motion animation, and fake music reactions.".
  + **Validation:** Quiz works, aura meter animates, DJ cards are shown. Ensure quiz is not gating overall experience.
* **ShuffleFeed - Remix Buttons (Prompt 3, ShuffleFeed.tsx):**
  + **Prompt:** "Upgrade the ShuffleFeed.tsx from a passive grid into an interactive PLUR-based remix dojo... For each feed item, summon a RemixModal with pitch/tempo sliders. Click Generate returns a new stub .mp3 URL (call /api/remix/mock). Update feed item's audio source to remix URL and tag '🔀 Remixed'.".
  + **Validation:** Modal opens, sliders change JSON payload, after "Generate" feed item plays new URL (even if mocked). No real playback or customization, just button styling.
* **Festival Voting Stage - Turntables & Audio (Prompt 5, FestivalVotingStage.tsx):**
  + **Prompt:** "Transform FestivalVotingStage.tsx into an interactive virtual rave stage... Add 2 glowing turntable decks (drag to scratch, click to echo/reverb, LED ring pulse). Place 'Hype the Crowd' button... Play a looping local .mp3 beat. Add mute/unmute toggle. Pulse background gradients, leaderboard bars, stage glow to beat. DJ voting cards show 'Preview Track' button (fakes switch). When vote, bar fills, DJ avatar glows, 'LIVE' label, confetti/pyro. Floating top banner 'Festival Energy'. Fake 'crowd chat' sidebar. Background: festival stage silhouette + subtle lasers. Everything sound-reactive, archetype-themed, PLUR hype.".
  + **Validation:** Stage background, DJ cards, voting counts, scratch/echo simulation work. **Audio must actually play**, decks animate and influence *simulated* music, votes should affect some state. Ensure sound, staging, and energy burst feel like a rave opening.
* **User Authentication (Core Task, not a single prompt):**
  + **Prompt:** "Create a fully working user auth flow for EDM Shuffle using Supabase (or recommended modern backend). Must support sign up, login, email/password reset, and user profile. UI responsive, branded, linked to app's state/context. All returned data must be real; do not use mocks. Add OAuth login... Ensure user-friendly error messages.".
  + **Validation:** Successful user registration, login, profile display with real Supabase data. Functional password reset and OAuth integration.
* **Web Audio API Engine (Core Task, not a single prompt):**
  + **Prompt:** "Build a fully functional DJ mix engine in React/TypeScript using the Web Audio API. Support two decks (A/B), play/pause, volume, crossfade, pitch, and echo controls. No HTML5 Audio fallback. All controls must be visible, interactive, and reflect actual audio state. Add a live waveform visualizer and BPM display...".
  + **Validation:** All deck controls functional and reflect actual audio state. Waveform visualizer uses real audio data and animates in sync. Cross-browser compatibility.

#### **5. Feature & Gap Analysis**

This table provides a concise overview of the current state of EDM Shuffle's features compared to the fully envisioned application, based on the latest user review and code audit.

| **Feature/Area** | **Current State (per your review & code)** | **Full Vision (as documented)** | **Progress Status** | **Key Gaps / Next Steps** |
| --- | --- | --- | --- | --- |
| **Landing Page & Branding** | Visuals, CTA for archetype quiz, static trending clips/gear drops. | Dynamic news, festival countdowns, team, authentic trending content (music, dance, fashion). | 60% (Visuals) | **Needs real data feeds and interactivity for trending content**. Equalizer bars not synced to sound. PLUR orbs lack function. |
| **Archetype Quiz & Flow** | Modal picker, basic quiz, result page. Choice is possible, but backend impact unclear. Quiz is hidden in bottom right, odd UX. | Fully interactive, nuanced, impacts site theme/user experience; unlocks special content; immersive "rave-baptism". | 40% (Shallow Logic) | **No connection to feed logic; quiz is still gating, not enhancing**. Archetype selection needs to drive personalized content and site theme. |
| **Challenge Feed (PLUR/Quests)** | Visual only, static fake cards, non-functional "Remix This/Play/Share" buttons. PLURstreak/crew features are visual-only. | Real, rotating dance/music challenges, progress tracking, unlocks, crew integration, reward/level system. "Remix This" should open modal with pitch/tempo sliders. | 10% (Placeholders) | **No feed content, remix action is just buttons—no real playback or customization**. Needs backend for dynamic challenges and user submissions. |
| **Marketplace** | Static product cards, looks visually fine. | Stripe integration, real inventory, digital AND physical item support, user trades, item history, archetype-specific gear, AI recommendations. | 15% (UI Exists) | **No e-commerce (Stripe), no cart, no checkout**. Items are fake, not clickable. No user upload for resale. |
| **DJ Mix Station** | Decks look interactive, all fake/simulated audio controls. Debug HUD present. | Full Web Audio API, track upload/playlist, real effects, mixing, save/share sets, playlist library. | 20% (Core UI) | **No real audio playback/mixing on decks**. Waveforms are visual only, not data-driven. Pitch/Echo effects are visual not audible. |
| **Festival Voting Stage** | Visual voting, no real competition. Voting button works but can be spammed. Turntables are static/visual only. | Scheduled/real-time festival events, DJ leaderboard, capped/validated voting, preview sets, rotating lineups. Interactive turntables with real (simulated) sound effects. | 20% (Visual Voting) | **Audio never plays**. Decks animate but don't influence music. Votes do not affect anything beyond count, no real headliner logic. |
| **Virtual Festival & Schedules** | None; not started. | Weekly/monthly festival calendar, lineup builder, integration with live streams, time zone support. | 0% | **Needs full design/dev**. |
| **Leaderboard/Headliners** | Visual leaderboard only, not live. | Real leaderboard: dynamic, based on voting/challenges, auto-headliner promotion, stats. | 10% (Just UI) | **Needs backend logic to populate and update dynamically**. |
| **Archetype Games/Avatar Crew** | Quiz repeats, some avatars shown, but not interactive. | Gamified squad/crew system, actual recruitment, member actions, inter-squad events, evolvable avatars. | 15% (Placeholders) | **Placeholder, not interactive**; no real crew building or game mechanics. |
| **Community/Chat** | Fake scrolling comments. | Real chat, private/group channels, fam/squad boards, in-app messaging, moderation. | 0% | **Not started.** |
| **PLUR Marketplace/Trading** | No trading, only gear preview. | Secure user-to-user trades, rarity, collectibles, leaderboards for traders. | 0% | **Not started.** |
| **User Auth/Profiles** | No signup/login. | Supabase auth, persistent profiles, social login, data sync, custom avatars. | 0% | **Not started.** |
| **Stripe & E-commerce** | None. | Live cart, checkout, digital/physical item fulfillment, payment receipts. | 0% | **Not started.** |
| **Mobile Responsiveness** | Looks decent, untested on all devices. | Full QA on iOS/Android, gesture support, touch-optimized controls. | ~50% (Untested) | **Needs thorough testing and optimization.** |
| **API Keys & Integrations** | No actual API keys implemented, only discussed. | Spotify (library/playlists, no direct audio), YouTube (content, no direct audio), Web Audio API, Supabase, Stripe, n8n, Printify, ElevenLabs, SDXL, Stability, Replicate. | Discussed Only | **No integrations are currently live for data fetching or external services.** |

### **EDM Shuffle Development Manual: Master Blueprint**

### **1. Executive Summary & Vision**

**EDM Shuffle** is a next-generation digital festival, DJ, and shuffle dance experience built to unite the global EDM community in a vibrant, interactive, and trustworthy platform.

Our goal: **Build the web’s most engaging digital rave, where talent rises, creativity is rewarded, and every participant feels seen, heard, and inspired**.

**Vision:** EDM Shuffle exists to **empower talent, celebrate diversity, and create new avenues for music lovers and creators**. We are building for fairness, transparency, and pure digital joy. All code is rigorously documented and built with multi-LLM collaboration at its core.

**Guiding Values & Priorities:**

* **Transparency** in progress, claims, and deliverables.
* **User empowerment** and creativity.
* **Modularity** and open standards.
* **No placeholder code** without explicit labeling and documentation.
* **Functionality over fantasy**.
* **AI + automation from day 1**.
* **Deep personalization**.
* **Archetype-based UI and content**.

### **2. Documentation Index**

This section lists all official project documentation files residing in the root directory. This index ensures all contributors (human or LLM) have immediate access to the project's canonical information.

* CLAUDE\_INTEGRITY\_RULES.md
* IMPLEMENTATION\_MICROTASKS.md
* PRD.md
* FRS.md
* MASTERPLAN.md
* claudeupdate.md
* AUDIO\_UI\_COMPONENTS.md
* README.md
* project\_overview.md
* technical\_architecture.md
* UI\_UX\_STRUCTURE.md
* CrowdFXLayer.md
* SubscribeModal.md
* FEATURE\_REALITY\_MATRIX.md (New document, critical for current state)
* USER JOURNEY MAP.md (New document, critical for current state)

### **3. PRD (Product Requirements Document)**

This Product Requirements Document (PRD) defines the high-level product vision, business goals, target audience, and the scope for the Minimum Viable Product (MVP) and beyond, with a strict focus on functional, verifiable features.

**1. Introduction**

* **Product Name:** EDM Shuffle
* **Vision:** To be the web’s most engaging digital rave experience, empowering talent, celebrating diversity, and fostering genuine connection through EDM, shuffle dance, and festival culture.
* **Problem:** Existing platforms are either static, lack deep interaction, or fail to combine music, dance, and community in an authentic, high-energy, and transparent way. Prior attempts at EDM Shuffle suffered from simulated/fake features.
* **Solution:** An AI-powered, multi-agent (Claude Code, Gemini CLI, Roo Code) platform that delivers real-time interactive DJ experiences, personalized archetype journeys, a peer-driven fashion marketplace, and a vibrant, authentic community hub.

**2. Goals**

* **Core Goal:** Deliver tangible, verifiable functionality over elaborate, simulated features.
* **MVP Goal:** Establish a fully functional user journey for core experiences (login, music playback, basic interaction/voting) that proves the core value proposition.
* **Monetization Goal:** Lay the foundation for future revenue streams (marketplace, premium features) based on real user engagement.
* **Integrity Goal:** Maintain absolute transparency regarding feature status (real vs. simulated/placeholder) throughout development.

**3. Target Audience**

* **Primary:** Gen Z EDM enthusiasts, shuffle dancers, aspiring DJs, festival-goers, ravewear creators/sellers.
* **Secondary:** Music producers, event organizers, dance instructors, fashion brands.

**4. Core Features**

* **User Authentication & Profiles:** Secure registration, login, profile management, archetype selection.
* **DJ Mix Station:** Interactive dual decks, real-time audio playback, crossfading, pitch control, echo FX, dynamic waveform visualization.
* **Festival Voting Stage:** Live DJ competitions, user voting, dynamic leaderboards, headliner selection.
* **Shuffle Feed & Challenges:** Dynamic feed of user-submitted dance content, archetype-specific challenges, gamified progression.
* **Community & Crews:** Live chat, private squad/fam groups, commenting, avatar/crew management, user-to-user interaction.
* **Fashion Marketplace:** Peer-to-peer resale for ravewear, affiliate listings, AI style matching, secure payment processing.
* **Autonomous Agent System:** AI-driven content curation, personalization, moderation, and operational automation.
* **Admin Dashboard:** Comprehensive tools for content, user, event, and product management.
* **Built-in Marketing Framework:** In-app campaign creation, content generation, and automation handoff to external platforms.

**5. Non-Functional Requirements**

* **Performance:** Fast load times, smooth animations (60 FPS target), responsive UI.
* **Scalability:** Designed for high concurrency (e.g., live festival events, thousands of users).
* **Security:** Robust user authentication (Supabase), secure API endpoints, protection against common web vulnerabilities.
* **Maintainability:** Modular component architecture, clean code, comprehensive documentation.
* **Accessibility:** WCAG 2.1 AA compliance where applicable (keyboard navigation, screen reader support, clear contrasts).
* **Truthfulness:** **NO FAKED LOGIC OR SIMULATED FEATURES WILL BE PRESENTED AS REAL.** Any placeholder or non-functional element must be explicitly labeled in the UI and code.

**6. Release Phases (High-Level)**

* **MVP (Initial Launch):** Focus on core user auth, basic real audio playback, archetype selection, and visible (but not necessarily fully backend-integrated) core loops.
* **Phase 2: Core Platform Enhancement:** Deepen audio features, integrate voting, begin real content agents.
* **Phase 3: Community & Monetization:** Implement social features, activate marketplace, advanced AI personalization.

**7. Testing & Validation Protocols**

* **Automated Testing:** Unit, integration, and end-to-end tests (Jest/Playwright).
* **Manual QA:** User flow tests, visual regression checks, cross-browser compatibility.
* **LLM Validation:** Require LLMs (Claude Code, Gemini CLI, Roo Code) to provide console logs, screenshots, and explicit confirmation of functionality before marking tasks complete.
* **Truth Audits:** Regular human review to compare claimed progress against actual, visible, and functional outputs.

**8. Future Considerations**

* Mobile native apps
* Advanced AI music generation
* VR/AR integrations for festivals
* Deeper blockchain/NFT integration

### **4. FRS (Functional Requirements Specification)**

This Functional Requirements Specification (FRS) provides a granular breakdown of *exactly what each feature must do*, including inputs, outputs, component dependencies, integration points, and defined failure states. It directly guides LLM development by outlining precise, verifiable expectations.

**1. User Authentication & Account System**

* **1.1 User Registration (Email/Password)**
  + **Description:** Allow new users to create an account via email and password.
  + **Inputs:** Email (string, valid format), Password (string, min 8 chars, strong), Password Confirm.
  + **Outputs:** User session token, redirect to onboarding/profile.
  + **Dependencies:** Supabase Auth, client-side validation.
  + **Failure States:** Invalid email format, password mismatch, email already exists, network error. Display specific error messages.
  + **Truth & Validation:**
    - **Prompt for Claude Code:** "Implement a secure, Supabase-backed user registration flow. Include client-side validation for email/password strength. On successful registration, log the user in and redirect to /onboarding. Handle and display specific error messages for invalid input, existing email, or network issues."
    - **Expected Output/Validation:**
      * Code for registration form component (AuthRegister.tsx).
      * Screenshots/video of successful registration and redirection.
      * Logs showing Supabase user creation.
      * Demonstration of error messages for all failure states.
      * **No fake success messages or redirects without actual user creation.**
* **1.2 User Login (Email/Password)**
  + **Description:** Allow existing users to log into their account.
  + **Inputs:** Email (string), Password (string).
  + **Outputs:** User session token, redirect to dashboard/home.
  + **Dependencies:** Supabase Auth.
  + **Failure States:** Invalid credentials, user not found, network error.
  + **Truth & Validation:**
    - **Prompt for Claude Code:** "Implement a login form for Supabase-backed authentication. On successful login, persist the session and redirect to /dashboard. Provide clear error messages for incorrect credentials or network issues. Ensure UI is responsive and accessible."
    - **Expected Output/Validation:**
      * Code for login form component (AuthLogin.tsx).
      * Screenshots/video of successful login.
      * Logs showing successful Supabase session creation.
      * Demonstration of error messages for invalid credentials.
      * Verify persistent session across browser refreshes.
* **1.3 Password Reset (Email)**
  + **Description:** Allow users to reset their password via an email link.
  + **Inputs:** Email (string).
  + **Outputs:** Confirmation message, email sent.
  + **Dependencies:** Supabase Auth (password reset functionality), email service integration (e.g., SendGrid configured via Supabase).
  + **Failure States:** Email not found, network error.
  + **Truth & Validation:**
    - **Prompt for Claude Code:** "Implement the password reset initiation flow. User enters email, receives confirmation. Supabase should send a password reset email. Do not mock email sending; verify Supabase integration for this specific feature."
    - **Expected Output/Validation:**
      * Code for password reset request component (PasswordResetRequest.tsx).
      * Confirmation message displayed in UI.
      * Logs confirming Supabase password reset email trigger.
      * **Proof:** If possible, confirm a test email is sent (e.g., via Supabase logs or a real test email client).
* **1.4 Profile Management (Placeholder-Free)**
  + **Description:** Allow authenticated users to view and update their profile information.
  + **Inputs:** Username (string, unique), Avatar URL (string), Archetype (enum), Bio (text).
  + **Outputs:** Updated user profile in database.
  + **Dependencies:** Supabase Database (User Profiles table), Supabase Storage (for avatars), user session.
  + **Failure States:** Duplicate username, invalid input, database update error.
  + **Truth & Validation:**
    - **Prompt for Claude Code:** "Build a user profile view/edit component (UserProfile.tsx). User can update username, avatar, and bio. Changes must persist to Supabase database. Show loading states during update. Ensure no placeholder data is used for user info."
    - **Expected Output/Validation:**
      * Code for profile component.
      * Screenshots/video of successful profile updates.
      * Supabase database logs showing data changes.
      * Demonstration of error handling for duplicate usernames or invalid inputs.

**2. Audio Engine & Mix Station**

* **2.1 True Web Audio API Engine**
  + **Description:** Implement a robust, multi-deck audio engine using the Web Audio API for real-time mixing.
  + **Inputs:** Audio file URLs, user control inputs (play, pause, volume, pitch, FX).
  + **Outputs:** Audibly mixed audio output, real-time audio visualization data.
  + **Dependencies:** Web Audio API, Howler.js (for simplified audio management, if chosen).
  + **Failure States:** Audio file 404, Web Audio API not supported, decoding errors. Display "Simulation Mode" badge if real audio fails.
  + **Truth & Validation:**
    - **Prompt for Claude Code:** "Implement a core audio context provider (useAudioEngine.ts). This must instantiate a real Web Audio API context. Create two independent audio sources (Deck A & B) that can load and play local .mp3 files. Expose play(), pause(), setTrack(url), setVolume() for each deck. Ensure cross-browser compatibility (Chrome, Edge, Safari, Firefox). Display a 'SIMULATION MODE' badge if the real Web Audio API fails to initialize or play sound. **DO NOT USE PLACEHOLDER AUDIO.**"
    - **Expected Output/Validation:**
      * Core audio engine code.
      * Logs confirming Web Audio API context creation.
      * Video demonstrating actual, audible playback of independent tracks on Deck A and B.
      * Screenshot of "SIMULATION MODE" badge if audio fails (e.g., by intentionally blocking audio load).
* **2.2 Deck A/B Real Playback, Crossfade, Pitch, Echo FX**
  + **Description:** Enable full interactive control over two DJ decks, including audio crossfading, pitch shifting, and echo effects.
  + **Inputs:** User interactions with sliders/buttons.
  + **Outputs:** Real-time audible changes to the mix.
  + **Dependencies:** Web Audio API nodes (GainNode, DelayNode, BiquadFilterNode), UI components (sliders, buttons).
  + **Failure States:** UI not affecting audio, effects not audible.
  + **Truth & Validation:**
    - **Prompt for Claude Code:** "Integrate UI controls (sliders for volume, pitch; buttons for echo, play/pause) with the useAudioEngine.ts from 2.1. Ensure the crossfader audibly blends between Deck A and B. Pitch controls must alter playback rate with an audible effect. Echo FX must produce a real delay/reverb sound. All controls must provide immediate visual feedback (e.g., slider position, button glow) that syncs to audible changes. No visual-only controls."
    - **Expected Output/Validation:**
      * Code for DJDeck.tsx and Crossfader.tsx.
      * Video demonstrating audible crossfading, pitch changes, and echo effects.
      * Video demonstrating visual feedback on controls precisely matching audio output.
      * **Ensure no hardcoded values for effects; all should be dynamic.**
* **2.3 Real-time Waveform + BPM Visualization**
  + **Description:** Visually represent the audio waveform and synchronize animations with the track's BPM.
  + **Inputs:** Audio buffer data, BPM detection.
  + **Outputs:** Animated waveform bars, UI elements pulsing to BPM.
  + **Dependencies:** Web Audio API (AnalyserNode), BPM detection library (if possible, or mock for now with explicit labeling).
  + **Failure States:** Static visualizations, animations not synced.
  + **Truth & Validation:**
    - **Prompt for Claude Code:** "Implement a real-time waveform visualizer using AnalyserNode, displaying frequency data as animated bars. Integrate a BPM display that reacts to the audio (or mock it with a clear 'MOCKED BPM' label if real-time detection is out of scope). Ensure all visual effects (e.g., glowing equalizer, floating sneakers, PLUR orbs) pulse or animate in sync with the BPM."
    - **Expected Output/Validation:**
      * Code for WaveformVisualizer.tsx and BpmAura.tsx.
      * Video showing dynamic, real-time waveform animation.
      * Video demonstrating UI elements (e.g., GlowCard, NeonButton, EqualizerBars) visibly reacting to BPM.
      * If BPM is mocked, clear // MOCKED BPM comments in code and a visible label in the UI.

**3. Voting & Festival Scheduling**

* **3.1 Real Voting Logic (Rate-Limited, Unique per User)**
  + **Description:** Implement a robust voting system for DJ competitions or other events.
  + **Inputs:** User ID, DJ/Candidate ID, Vote action.
  + **Outputs:** Vote recorded, real-time vote count update.
  + **Dependencies:** Supabase Database (Votes table), Supabase Auth (for user ID), API endpoints for voting.
  + **Failure States:** Duplicate vote, unauthorized vote, network error. Implement rate-limiting.
  + **Truth & Validation:**
    - **Prompt for Claude Code:** "Implement backend and frontend for real-time voting on the Festival Voting Stage. Votes must be unique per authenticated user (rate-limited to 1 vote/minute/user). Display real-time vote counts on DJ cards. Handle and display errors for duplicate votes or network issues. Ensure no placeholder votes."
    - **Expected Output/Validation:**
      * Code for voting API endpoint and VotingButton.tsx.
      * Logs confirming Supabase vote writes.
      * Video showing real-time vote count updates and rate-limiting enforcement.
      * Attempt to vote without auth/too frequently, verify error message.
* **3.2 Festival Schedule UI + Backend Data Model**
  + **Description:** Display an interactive festival schedule with events, artists, and stages.
  + **Inputs:** Admin input for events.
  + **Outputs:** Festival schedule displayed, event details.
  + **Dependencies:** Supabase Database (FestivalEvents, Artists, Stages tables), API for fetching schedule.
  + **Failure States:** Empty schedule, data load error.
  + **Truth & Validation:**
    - **Prompt for Claude Code:** "Build a backend data model (Supabase tables) and API for festival schedules. Events should include: name, date, time, stage, artists, genre, link. Implement a responsive UI to display this schedule. Data must come from Supabase, not hardcoded. Show skeleton loading states and 'No events' fallback."
    - **Expected Output/Validation:**
      * Supabase schema and API code.
      * UI for FestivalSchedule.tsx displaying real data.
      * Screenshot of skeleton loading and empty state.
* **3.3 Event Creation/Joining, Headliner Logic, Live Leaderboard**
  + **Description:** Allow users to RSVP/join events, crown headliners based on votes, and display a live leaderboard.
  + **Inputs:** User RSVP, vote counts.
  + **Outputs:** RSVP recorded, headliner announced, updated leaderboard.
  + **Dependencies:** Supabase (UserEvents, Leaderboard tables), voting system.
  + **Truth & Validation:**
    - **Prompt for Claude Code:** "Implement RSVP functionality for festival events. Users can join/unjoin. Based on voting data (from 3.1), implement headliner logic: when a DJ's votes exceed a threshold (e.g., 5000), they are marked as the headliner for that event. Display a live, animating leaderboard that updates in real-time with DJ vote counts. Headliner announcement should trigger confetti and a 'stinger' sound effect. No fake leaderboards or announcements."
    - **Expected Output/Validation:**
      * Code for RSVPButton.tsx, FestivalLeaderboard.tsx, HeadlinerConfetti.tsx, HeadlinerStinger.tsx.
      * Video showing RSVP flow and real-time leaderboard updates.
      * Video demonstrating confetti/stinger on headliner threshold met.
      * Confirm vote counts trigger accurate headliner selection.

**4. Challenges & Social**

* **4.1 Challenge Feed – Backend/API & Frontend Integration**
  + **Objective:** Deliver a robust, dynamic challenge feed system with reliable data handling, accessibility, and bulletproof error/fallback states.
  + **Tasks:**
    - **Backend:** Create challenges table (fields: id, title, description, archetype, bpm, creatorId, type, status, mediaURL, submissionCount, start/end, etc.). Implement API endpoints for listing and retrieving challenges (paginated, filterable). Add validation for all inputs; strictly reject malformed/oversized data. Enforce rate limits for challenge creation (e.g., 3/hour/user).
    - **Frontend:** Fetch and render challenge list with skeleton loading states and paginated/infinite scroll UI. Show “No challenges available” or explicit error/fallback state if fetch fails (never blank). Accessible, mobile-first card layout. All data strictly typed; undefined/null always guarded. If in development, clearly label/test placeholder/demo data in UI and code comments. Retry/reload button and error context for failed requests. Filter/sort controls by archetype, difficulty, etc.
  + **Validation & Failsafe:** Simulate API offline or empty data: UI must show “No challenges found” + retry option. Try loading page on slow connection: must show skeleton, never unstyled blank. Code review: Confirm no placeholder/mock code ships to prod without explicit // TODO:/// NOTE: and fallback path.
  + **Prompt for Claude Code:** "Create a complete backend and API for the EDM Shuffle challenge feed, including a challenges Supabase table with validated fields. Implement API endpoints for paginated and filterable challenge lists. Frontend should fetch and render these dynamically with skeleton loading, pagination, and robust error handling (never blank screen). Ensure all data is strictly typed and mobile-first. No static data or placeholders."
  + **Expected Output/Validation:**
    - Supabase schema + API routes.
    - ChallengeFeed.tsx component code.
    - Screenshots/video of dynamic loading, pagination, error states.
    - Logs showing successful API calls to Supabase.
* **4.2 Challenge Submission Flow**
  + **Objective:** Build a reliable, secure, and user-friendly challenge submission system for files and metadata.
  + **Tasks:**
    - **Backend:** Create submissions table (fields: id, challengeId, userId, caption, mediaUrl, tags, timestamp, status). Implement API endpoint for submission creation. File storage: secure upload to Supabase Storage. Implement robust file validation (type, size, malicious content scan), and rate-limit submissions.
    - **Frontend:** User must be authenticated. Form accepts media file (mp4/webm/jpg/png/gif, max 30MB), caption, squad tags. Client validates all fields before upload (disable submit if invalid). Show progress indicator during upload, error on fail, and success animation on confirmation. Allow user to retry after failure (don’t clear form on error).
  + **Validation & Failsafe:** Attempt large file upload (must fail gracefully). Try submitting without required fields (must show validation errors). Test network interrupt during upload (must show retry/error). Verify submitted content appears in a review queue (admin only).
  + **Prompt for Claude Code:** "Create a secure, fully validated challenge submission flow for EDM Shuffle. Frontend: User must be authenticated; form accepts media file (max 30MB, standard formats), caption, squad tags. Client validates fields before upload. Backend: Supabase submissions table, API endpoint for creation. Secure file upload to Supabase Storage with type/size validation and rate-limiting. Show accurate upload progress, success, and retryable error states. No mock flows."
  + **Expected Output/Validation:**
    - Supabase schema + API routes for submissions.
    - ChallengeSubmissionForm.tsx component code.
    - Video demonstrating full submission flow (upload, success, error).
    - Verify file upload to Supabase Storage.
* **4.3 PLUR Leaderboard & Gamification**
  + **Objective:** Integrate a real-time leaderboard for "PLUR" points and challenge winners, driving engagement.
  + **Tasks:**
    - **Backend:** Create user\_scores table (id, userId, totalPlur, challengesWon, streaks, badges). API endpoints for updating scores (triggered by challenge submission approval, voting, etc.) and fetching leaderboard data (paginated, sortable).
    - **Frontend:** Display live global and challenge-specific leaderboards. Show PLUR streak tracking (top-left counter) and PLUR meter progression with emoji. Implement animations for rank changes or new achievements.
  + **Validation & Failsafe:** Simulate score updates (verify real-time rank changes). Test edge cases (tie scores, very high scores). Ensure leaderboards degrade gracefully if backend fails.
  + **Prompt for Claude Code:** "Build a real-time PLUR leaderboard. Backend: Supabase user\_scores table to track PLUR points, challenge wins, streaks. API endpoints to update scores and fetch leaderboards. Frontend: Display global and challenge-specific leaderboards. Implement a visible PLUR streak counter and a PLUR meter with emoji progression (🫂❤️🌀💥) that increases with interactions (e.g., successful challenge submissions). Animations for rank changes are crucial. No static leaderboards."
  + **Expected Output/Validation:**
    - Supabase schema + API routes for scores.
    - Leaderboard.tsx and PLURstreakMeter.tsx components.
    - Video demonstrating real-time score updates and animated leaderboards.
    - Verify PLUR streak/meter updates on simulated interactions.

**5. Marketplace**

* **5.1 Stripe/Payments Integration**
  + **Objective:** Integrate Stripe (or equivalent) for secure and real payment processing.
  + **Tasks:**
    - **Backend:** Set up Stripe webhooks for payment events (success, failure, refunds). Implement API endpoints for:
      * Creating Stripe Checkout Sessions for products/tickets.
      * Handling successful payments and updating order status.
    - **Frontend:** Display "Buy Now" button. Redirect to Stripe Checkout for payment. Handle success/failure redirects from Stripe.
  + **Validation & Failsafe:** Test with Stripe test keys (success, failed, cancelled payments). Ensure error messages are clear and secure (no sensitive info). Implement retry logic for payment failures.
  + **Prompt for Claude Code:** "Integrate Stripe for product purchases. Backend: API endpoint for creating Stripe Checkout Sessions. Set up Stripe webhooks to update order status post-payment. Frontend: 'Buy Now' buttons initiate Stripe Checkout. Handle success/cancellation redirects. Use Stripe test keys; prove successful payment processing to update a mock order status in Supabase. No fake payment flows."
  + **Expected Output/Validation:**
    - Backend Stripe integration code.
    - Frontend BuyButton.tsx and payment handling logic.
    - Video demonstrating a full Stripe test payment flow (from click to success/failure).
    - Logs confirming Stripe webhook calls.
* **5.2 Real Product Listings (No Placeholders)**
  + **Objective:** Enable dynamic, real product listings from a backend database.
  + **Tasks:**
    - **Backend:** Create products table (id, name, desc, price, image, archetype, stock, active). API endpoints: list (filter/sort), get by id, create/update/delete (admin only). Validate all data (types, prices, URLs, inventory).
    - **Frontend:** Fetch products live (never ship with static lists). Product cards: show all info, “Buy” or “Add to Cart” button (disable if out of stock). Handle fetch errors with visible fallback (“No products available. Retry.”).
  + **Validation & Failsafe:** Simulate empty product list (show fallback). Test filtering/sorting. Verify stock status affects UI. All admin/management UIs must be auth-protected. Real-time inventory check before each add-to-cart.
  + **Prompt for Claude Code:** "Build a fully dynamic, real product marketplace for EDM Shuffle. Backend: products Supabase table, API for listing (filterable/sortable) and retrieving products. Implement robust data validation. Frontend: Fetch products dynamically; display product cards with name, price, image. 'Buy'/'Add to Cart' buttons should be disabled if out of stock. Show fallback UI on fetch errors (never a blank screen). No hardcoded product lists or placeholder images in production."
  + **Expected Output/Validation:**
    - Supabase schema + API routes for products.
    - ProductCard.tsx and Marketplace.tsx components.
    - Video demonstrating dynamic product loading, filtering (if implemented), and out-of-stock behavior.
    - Verify dynamic content from Supabase.
* **5.3 Order Confirmation & Digital Download/Gift Flows**
  + **Objective:** Provide clear order confirmation and manage digital product delivery.
  + **Tasks:**
    - **Backend:** Upon successful payment, create orders entry (userId, productId, total, status, timestamp) and digital\_assets entry (userId, assetUrl, expiration). Implement API endpoint for digital asset delivery/download (secure, time-limited links).
    - **Frontend:** Display order confirmation page/modal with unique order ID and purchased items. For digital goods, provide a direct download link or access to a "My Downloads" section.
  + **Validation & Failsafe:** Simulate successful digital purchase (verify download link appears). Test invalid/expired download link (must show error). Ensure robust payment failure messaging.
  + **Prompt for Claude Code:** "Implement order confirmation and digital product delivery. Backend: orders Supabase table upon successful Stripe payment. For digital products, create a digital\_assets entry with a secure, time-limited download URL. Frontend: Display a confirmation page/modal with order details. For digital goods, provide a functional (test) download link. Ensure error handling for failed downloads/access. No fake confirmations or download links."
  + **Expected Output/Validation:**
    - Supabase schema updates and API for orders/digital assets.
    - OrderConfirmation.tsx component.
    - Video demonstrating successful order confirmation and a test digital download.
    - Test secure link behavior (e.g., trying to access without purchase).

**6. Community & Crews**

* **6.1 Live Chat, Squad/Fam Features, and Commenting**
  + **Objective:** Enable real-time user-to-user and group communication, supporting general chat, squad-specific chat, and commenting on content.
  + **Tasks:**
    - **Backend:** Supabase Realtime (for live chat) or similar WebSocket. messages table (senderId, receiverId/groupId, content, timestamp). comments table (userId, contentId, content, timestamp). API endpoints for sending/retrieving messages/comments, and fetching chat history.
    - **Frontend:** Real-time chat UI (global, squad-specific). Commenting section on challenge clips/artist profiles. Input validation (profanity filter, message length). Display send/read receipts (optional).
  + **Validation & Failsafe:** Test concurrent messages (no data loss). Simulate network disconnect (verify graceful reconnection/message queuing). Try sending profanity/long messages (verify filter/truncation).
  + **Prompt for Claude Code:** "Implement real-time live chat (global and squad-specific) and content commenting using Supabase Realtime. Backend: messages and comments Supabase tables with sender/receiver/content/timestamp. API endpoints for history. Frontend: Chat UI must display real messages and comments dynamically. Implement basic input validation (e.g., text length, light profanity filter). Verify real-time updates across multiple users. No static chat logs or comments."
  + **Expected Output/Validation:**
    - Supabase schema + API routes for chat/comments.
    - LiveChat.tsx and CommentsSection.tsx components.
    - Video demonstrating real-time message sending/receiving across multiple users.
    - Verify chat history persistence and content filtering.
* **6.2 Avatar, “Crew” Management, User-to-User Interactions**
  + **Objective:** Allow users to create/manage crews (squads/fams), invite/kick members, and enhance user-to-user interaction.
  + **Tasks:**
    - **Backend:** crews table (id, name, leaderId, description, privacy, memberCount, archetype). crew\_members join table (crewId, userId, role). API endpoints for:
      * Create/edit/delete crew.
      * Invite/accept/decline/kick member.
      * Change member role.
      * View crew profile/roster.
    - **Frontend:** Crew creation form. Crew dashboard (roster, invite system). User profiles display crew membership. Direct messaging/profile viewing.
  + **Validation & Failsafe:** Test inviting non-existent user. Try to join private crew without invite. Verify only leader can kick members. Simulate network errors during crew actions.
  + **Prompt for Claude Code:** "Implement crew/squad management. Backend: crews and crew\_members Supabase tables to store crew data and member relationships (leader, member roles). API endpoints for create/edit/delete crew, invite/accept/kick members. Frontend: Build CrewCreator.tsx and CrewDashboard.tsx. User profiles should display crew membership. Ensure all actions (invites, kicks) are authenticated and authorized (e.g., only leaders can kick). No dummy crews or static membership."
  + **Expected Output/Validation:**
    - Supabase schema + API routes for crews.
    - CrewCreator.tsx and CrewDashboard.tsx components.
    - Video demonstrating crew creation, inviting a test user, and accepting/declining invites.
    - Verify permission enforcement (e.g., regular member cannot kick).

**7. Admin + Docs**

* **7.1 Admin Dashboard for Event/Product/Content Management**
  + **Objective:** Provide a secure, centralized web interface for administrators to manage all platform content and users.
  + **Tasks:**
    - **Backend:** Implement role-based access control (admin, moderator, content manager). API endpoints for:
      * CRUD operations on FestivalEvents, Products, Challenges, Users, Crews, Comments, Reports.
      * View live activity feed and audit logs of admin actions.
    - **Frontend:** Secure login (with 2FA option). Dashboard UI with navigation for managing different entities. Data tables with sort, filter, search, pagination.
  + **Validation & Failsafe:** Attempt to access dashboard as regular user (blocked with clear message, logged attempt). Create, edit, delete entities (verify data updates and audit trail is updated). Simulate 2+ admins making changes simultaneously (test for consistency and no race conditions).
  + **Prompt for Claude Code:** "Develop a full-featured admin dashboard for EDM Shuffle. Implement role-based access (admin, moderator). Dashboard must allow management of: Festival schedule and event data (CRUD), Users (view, ban, promote/demote), Crew/squad approval, Challenges (approve, remove), Product listings (CRUD, inventory), Comments, and abuse reports. Include a live activity feed and audit logs for every admin action. Ensure secure login (no hardcoded credentials) and mobile-friendly UI. No dummy dashboards."
  + **Expected Output/Validation:**
    - Backend API for admin operations.
    - AdminDashboard.tsx and related sub-components.
    - Video demonstrating admin login and performing CRUD operations on test data (e.g., creating a new festival event, banning a user).
    - Logs confirming audit trail entries for admin actions.
* **7.2 Documentation Pipeline: Root Directory + Microtask Enforcement**
  + **Objective:** Ensure all project documentation remains up-to-date, versioned, and is a mandatory part of the development workflow.
  + **Tasks:**
    - **Pipeline Setup:** All project docs (PRD.md, FRS.md, MASTERPLAN.md, CLAUDE\_INTEGRITY\_RULES.md, etc.) live in project root. Add DOCUMENTATION\_INDEX.md for table of contents and version history.
    - **CI/CD Integration:** On each PR/merge/deployment, run a checklist: "Were all docs touched by this feature updated? Is a new microtask/checklist present for ongoing features? If any doc is not updated, CI fails and merge is blocked."
    - **Tooling:** Encourage use of Markdown linting/formatting tools (e.g., remark-lint).
  + **Validation & Failsafe:** Try deploying with a stale doc (CI should fail). Make a feature PR without a doc update (system should block and provide checklist feedback). Change doc, merge, and view version history (history should be complete, searchable, and accurate).
  + **Prompt for Claude Code:** "Implement an automated documentation pipeline. All project docs (PRD.md, FRS.md, MASTERPLAN.md, CLAUDE\_INTEGRITY\_RULES.md, etc.) must reside in the project root. Create DOCUMENTATION\_INDEX.md as a central TOC and version tracker. Configure CI/CD to enforce document updates: if a feature PR touches a module, relevant docs MUST be updated, or the PR is blocked. Use linting for Markdown quality."
  + **Expected Output/Validation:**
    - DOCUMENTATION\_INDEX.md content.
    - CI/CD configuration snippets (e.g., GitHub Actions, Netlify build hooks).
    - Demonstration of a PR failing due to missing doc updates.
* **7.3 Integrity & Compliance Automation**
  + **Objective:** Continuously check platform logic for compliance with CLAUDE\_INTEGRITY\_RULES.md and legal/policy requirements (e.g., GDPR, copyright, fair use).
  + **Tasks:**
    - **Automated Checks:** Run integrity rule tests before deploy (e.g., check for placeholder code, mock data, API key exposure). Validate consent and user data handling per privacy policy. Scan codebase for legal/compliance violations (using OSS tools or custom scripts).
    - **Reporting:** All failures produce actionable errors, not warnings.
  + **Validation & Failsafe:** Intentionally add a mock data leak or bad code (CI should block deploy). Remove a privacy consent feature (test should catch and fail). Run compliance script with no errors (confirm clean pass and clear logs).
  + **Prompt for Claude Code:** "Set up automated integrity/compliance checks. Configure pre-deploy hooks to scan for: undeclared placeholder code, exposed API keys, and violations of user consent/privacy. Implement basic data validation checks. All failures must halt deployment and provide actionable error messages. No silent failures."
  + **Expected Output/Validation:**
    - CI/CD configuration with integrity checks.
    - Code for custom integrity scripts (if needed).
    - Video demonstrating CI/CD failing when an integrity rule is violated.
* **7.4 Periodic Masterplan & FRS Review**
  + **Objective:** Ensure alignment between documentation, live code, and user feedback through scheduled reviews.
  + **Tasks:**
    - **Review Checklist:** After every major milestone, project leaders must review the FRS and MASTERPLAN against live code and user feedback. Checklist: "Every live feature accounted for in FRS? All known bugs or gaps logged and tracked? User feedback integrated into future phases?"
    - **Version Control:** Use versioned doc tracking in DOCUMENTATION\_INDEX.md.
    - **Remediation:** If discrepancies found, update docs or backlog, assign microtasks.
  + **Validation & Failsafe:** Compare doc to code and feature set (no undocumented features). Simulate review: find gap, update doc, reassign microtask. Pull random doc section (should match app and user experience exactly).
  + **Prompt for Claude Code:** "Implement a system to enforce periodic review of FRS and MASTERPLAN. Add a feature to DOCUMENTATION\_INDEX.md for version tracking of these core docs. Outline a process (e.g., a README section or an internal tool) to facilitate comparing current features against FRS/MASTERPLAN, logging discrepancies as microtasks. No 'drift' between docs and production."
  + **Expected Output/Validation:**
    - Updated DOCUMENTATION\_INDEX.md with versioning capabilities.
    - A Markdown template for review notes/checklists.
    - Demonstration of adding a discrepancy as a microtask.

**8. Built-in Marketing Framework**

* **8.1 Marketing Dashboard (Campaign Management & Automation Handoff)**
  + **Objective:** Provide an in-app dashboard for building and managing marketing campaigns, integrating with external automation tools.
  + **Tasks:**
    - **Backend:** Implement role-protected access. campaigns table (id, name, status, platforms, startDate, endDate, createdBy). Store content assets (copy, images, webhooks). API endpoints for campaign CRUD.
    - **Frontend:** Campaign creation wizard (select platforms: Instagram, TikTok, X, Threads, Facebook, YouTube, Printify, etc.). Live preview (mobile/desktop/social card view). Scheduling calendar for posts.
    - **Integration (n8n/Printify):** For every campaign, generate a webhook/JSON config for n8n to post/schedule/send content to each platform. Sync campaign assets, product descriptions, and drop schedules to Printify.
  + **Validation & Failsafe:** Create a test campaign (verify data persistence). Generate a test n8n webhook (verify structure and data). Simulate failed Printify sync (display error).
  + **Prompt for Claude Code:** "Develop a role-protected Marketing Dashboard within EDM Shuffle’s admin suite. Features: Campaign creation wizard (select platforms, define schedule). Content generation (LLM-driven copy/hashtags - *initiate via API calls, don't implement full LLM*). Output n8n webhooks/JSON config for automated posting. Sync with Printify for product drops. Ensure role-based access and logging of all actions. No dummy campaigns or non-functional integrations."
  + **Expected Output/Validation:**
    - Supabase schema + API for campaigns.
    - MarketingDashboard.tsx component.
    - Video demonstrating campaign creation and generation of n8n/Printify-compatible JSON/webhooks.
    - Logs confirming data storage and webhook generation.
* **8.2 Content Generation & Library**
  + **Objective:** Create a searchable, filterable library for all marketing content assets, with AI-driven suggestions for reuse.
  + **Tasks:**
    - **Backend:** content\_library table (id, type, assetUrl, textContent, tags, platform, campaignId, ownerId, lastUsed). API endpoints for CRUD and search.
    - **Frontend:** Internal content library module with quick search, preview, and copy/paste/share tools. AI suggestion: propose “next post” or “revive successful campaign” (via LLM API calls). Export to CSV, JSON, or direct n8n/Printify pipeline.
  + **Validation & Failsafe:** Create, save, and retrieve content from the library. Filter and reuse in new campaigns. Test export and direct pipeline push to n8n/Printify.
  + **Prompt for Claude Code:** "Build an internal content library module in the marketing dashboard. Backend: content\_library Supabase table to store marketing assets (copy, image URLs, tags). API for search, retrieval, and CRUD. Frontend: UI for managing the library. Users can create, save, and retrieve marketing content. Include AI suggestion (via *stubbed* API endpoint for LLM) to propose content reuse. Allow export to JSON. No hardcoded content."
  + **Expected Output/Validation:**
    - Supabase schema + API for content library.
    - ContentLibrary.tsx component.
    - Video demonstrating content creation, saving, searching, and previewing.
    - Logs confirming data persistence in Supabase.
* **8.3 Best Practices, Failsafes, & Innovation**
  + **Rate-limited/sandbox mode** for mass posting (avoids bans/duplication).
  + All campaign copy is human-editable and LLM-suggested, never forced live without approval.
  + Every automation connection logs all sent/received payloads.
  + Library auto-suggests content blocks based on seasonality, festival schedule, and product drops.
  + Roles/permissions enforced for all create/edit/publish/export.

### **5. Implementation Roadmap**

This roadmap outlines the chronological phases of EDM Shuffle's development, focusing on actionable, verifiable features. Each phase will adhere to strict anti-fabrication rules, and progress will be tracked via claudeupdate.md and FEATURE\_REALITY\_MATRIX.md.

* **Phase 1: User Authentication & Account System**
  + User Registration & Login (Supabase)
  + Email Password Reset
  + Profile Management (View/Update)
* **Phase 2: Audio Engine & Mix Station**
  + Implement true Web Audio API engine (no placeholders)
  + Enable Deck A/B real playback, crossfade, pitch, echo FX
  + Real-time waveform + BPM visualization synced to audio
  + Cross-browser testing (Chrome, Edge, Safari, Firefox)
* **Phase 3: Voting & Festival Scheduling**
  + Add real voting logic (rate-limited, unique per user)
  + Build festival schedule UI + backend data model
  + Implement event creation/joining, headliner logic, and live leaderboard
* **Phase 4: Challenges & Social**
  + Hook up challenge feed to backend (dynamic challenges)
  + Allow challenge submission (file upload, caption, squad tagging)
  + Integrate leaderboard for “PLUR” and challenge winners
* **Phase 5: Marketplace**
  + Add Stripe or other payments integration
  + Enable real product listings (no placeholders)
  + Order confirmation and digital download/gift flows
* **Phase 6: Community & Crews**
  + Implement live chat, squad/fam features, and commenting
  + Avatar, “crew” management, user-to-user interactions
* **Phase 7: Admin + Docs**
  + Build admin dashboard for event/product/content management
  + Keep all project docs up to date in root (DOCUMENTATION\_INDEX.md enforced)
  + Cross-check FRS and Masterplan after every milestone (Periodic review automation)
  + Integrity & Compliance Automation (placeholder detection, API key exposure, privacy rules)
* **Phase 8: Built-in Marketing Framework**
  + Marketing Dashboard (Campaign management & automation handoff to n8n/Printify)
  + Content Generation & Library (for re-usable marketing assets)
  + Best Practices, Failsafes, & Innovation (rate-limiting, approval workflows, logging)

### **6. Phase-by-phase Features/Prompts/Tasks**

(This section details each task with specific LLM prompts and validation steps. Due to the length, only a sample is shown here. The full content will be in the COMPREHENSIVE\_TASKS\_VALIDATION.md file, which these examples are drawn from.)

**Context & Memory Reminder for LLMs (to include at top of every new session):** "You are a developer for EDM Shuffle. Prioritize functionality over fantasy. All code must be real and verifiable. Any simulated, hardcoded, or placeholder features MUST be explicitly labeled with // TODO: Not implemented, // MOCK:, or // SIMULATION MODE: comments, and its state made obvious in the UI. Reference CLAUDE\_INTEGRITY\_RULES.md before starting. Consult IMPLEMENTATION\_MICROTASKS.md for ordering. Provide screenshots or console logs as proof of completion. If unable to complete a task, state the limitation and suggest an alternative tool (Gemini CLI, Roo Code) or human intervention. **NEVER fabricate progress.**"

#### **PHASE 1: User Authentication & Account System**

**1.1 User Registration & Login (Supabase)**

* **Task:** Implement user registration, email/password login, and persistent session using Supabase. All screens must be branded for EDM Shuffle, responsive, and return real user data.
* **Optimized Prompt for Claude Code:** "Implement a secure, Supabase-backed user registration and login flow for EDM Shuffle.
  + **Registration (/register):** Collect email and password (with client-side validation for format and strength). On success, create user in Supabase and auto-login.
  + **Login (/login):** Authenticate against Supabase.
  + **Session Management:** Persist user session across browser refreshes.
  + **UI:** Use neon aesthetic (Tailwind CSS, Framer Motion for subtle animations). Responsive for mobile/desktop.
  + **Error Handling:** Display clear, user-friendly error messages for invalid inputs, existing emails, incorrect credentials, or network issues.
  + **No Placeholders:** All user creation and authentication must be real, tied to Supabase.
  + **Files:** AuthRegister.tsx, AuthLogin.tsx, useAuth.ts (hook for Supabase integration).
  + **Validation Steps:**
    - **Screenshot/Video 1:** Successful new user registration and redirection to /dashboard.
    - **Screenshot/Video 2:** Successful login of an existing user.
    - **Console Logs:** Show Supabase auth.signUp and auth.signInWithPassword calls succeeding.
    - **Error Test:** Demonstrate invalid email, short password, and existing email errors.
    - **Refresh Test:** Log in, refresh browser, ensure user session persists."

**1.2 Email Password Reset (Supabase)**

* **Task:** Implement user password reset functionality via email.
* **Optimized Prompt for Claude Code:** "Build the password reset initiation and confirmation flow for EDM Shuffle.
  + **Request Form (/forgot-password):** User enters email to receive a password reset link.
  + **Backend:** Integrate with Supabase's auth.resetPasswordForEmail method. This should trigger a real email from Supabase's configured email service.
  + **Confirmation UI:** Display a clear message instructing the user to check their email.
  + **Reset Form (/reset-password):** After clicking the email link, user enters and confirms new password. This must use Supabase's auth.updateUser method.
  + **Error Handling:** Handle cases like invalid email, expired token, or network issues gracefully.
  + **No Mocking:** Prove that Supabase is genuinely sending the reset email (e.g., via Supabase logs or a test email inbox if possible).
  + **Files:** PasswordResetRequest.tsx, PasswordResetForm.tsx.
  + **Validation Steps:**
    - **Screenshot/Video 1:** User submits email on /forgot-password, sees confirmation message.
    - **Supabase Logs:** Provide log snippets confirming the auth.resetPasswordForEmail call and (if available) email sending event.
    - **Screenshot/Video 2:** User successfully resets password via the link and new form.
    - **Functional Test:** Attempt to log in with the old password (must fail) and then with the new password (must succeed)."

**1.3 User Profile Management (View & Update)**

* **Task:** Create a profile page where authenticated users can view and update their personal information (username, avatar, bio, archetype).
* **Optimized Prompt for Claude Code:** "Develop a fully functional user profile management page (/profile).
  + **Display:** Show current user's username, avatar, bio, and chosen archetype. Data must come directly from the Supabase profiles table.
  + **Edit Functionality:** Allow users to update their username, upload a new avatar (to Supabase Storage), and edit their bio.
  + **Archetype Display:** Show the selected archetype. **DO NOT** make it editable via this page yet, but ensure the UI element is ready for future integration.
  + **Persistence:** All changes must be written back to the Supabase profiles table.
  + **Loading/Feedback:** Show loading indicators during data fetch/save. Provide success/error notifications.
  + **Files:** UserProfile.tsx, useProfile.ts (hook for data fetch/update), optional AvatarUpload.tsx.
  + **Validation Steps:**
    - **Screenshot/Video 1:** User viewing their profile.
    - **Screenshot/Video 2:** User successfully updating their username and bio.
    - **Screenshot/Video 3:** User successfully uploading a new avatar.
    - **Supabase Logs:** Show supabase.from('profiles').update() calls with new data.
    - **Persistence Test:** Refresh the page after updates; verify data remains changed."

#### **PHASE 2: Audio Engine & Mix Station**

**2.1 Core Web Audio API Engine & Deck Architecture**

* **Task:** Implement a robust, multi-deck audio engine using the Web Audio API for real-time mixing, capable of playing independent tracks.
* **Optimized Prompt for Claude Code:** "Build the foundational Web Audio API engine for EDM Shuffle's DJ Mix Station.
  + **Core Context:** Create a global AudioContext provider.
  + **Independent Decks:** Implement two independent audio sources (DeckA, DeckB) that can load and play *separate local MP3 files*. Each deck should have its own AudioBufferSourceNode connected to the AudioContext.
  + **Controls:** Expose basic play(), pause(), loadTrack(url) methods for each deck.
  + **Error Handling:** If Web Audio API is not supported or a track fails to load, display a visible 'SIMULATION MODE' badge on the UI.
  + **No Placeholders:** All audio playback must be real via Web Audio API. Do not hardcode track URLs that are not actual, playable files.
  + **Files:** useAudioEngine.ts (main hook), DeckA.tsx, DeckB.tsx.
  + **Validation Steps:**
    - **Video 1:** Load /dj-mix page. Play a track on Deck A, then a different track on Deck B simultaneously. Verify both play audibly and independently.
    - **Console Logs:** Show AudioContext and AudioBufferSourceNode instances being created. Log successful track loading for each deck.
    - **Error Test:** Intentionally provide a bad track URL or disable Web Audio API (if browser allows) and screenshot/video the 'SIMULATION MODE' badge."

**2.2 Real-time Crossfader, Pitch & Echo FX**

* **Task:** Enable full interactive control over two DJ decks, including audio crossfading, pitch shifting, and echo effects, with immediate audible and visual feedback.
* **Optimized Prompt for Claude Code:** "Integrate advanced audio controls and effects into the DJ Mix Station.
  + **Crossfader:** Implement a functional crossfader (Crossfader.tsx) that audibly blends volume between DeckA and DeckB using GainNodes. Visual feedback (e.g., percentage, active deck highlight) must sync accurately to audio output.
  + **Pitch Control:** Add pitch sliders/controls that *audibly alter the playback rate* of each deck's track. Visual indicators should show the current pitch value.
  + **Echo FX:** Implement an audible echo effect for each deck using DelayNodes in the Web Audio API. An 'Echo' button or slider should visibly toggle/control the effect.
  + **Volume Controls:** Ensure individual volume sliders for DeckA and DeckB *audibly* adjust the gain.
  + **Visual-Audio Sync:** All controls (crossfader, pitch, echo, volume) must provide immediate visual feedback (e.g., slider position, button glow, pulsing elements) that *directly reflects the audible changes*. No visual-only interactions.
  + **Files:** Crossfader.tsx, PitchControl.tsx, EchoFX.tsx (or integrate into DJDeck.tsx).
  + **Validation Steps:**
    - **Video 1:** Demonstrate smooth, audible crossfading from Deck A to B and vice-versa.
    - **Video 2:** Play a track and adjust pitch; show audible change and visual pitch indicator updating.
    - **Video 3:** Activate echo FX and demonstrate the audible delay effect.
    - **Console Logs:** Log GainNode and DelayNode value changes, ensuring they correspond to UI actions."

**2.3 Real-time Waveform + BPM Visualization**

* **Task:** Display an animating waveform visualizer and synchronize UI elements to the track's BPM.
* **Optimized Prompt for Claude Code:** "Implement real-time audio visualization and BPM syncing for the DJ Mix Station.
  + **Waveform Visualizer:** Create a visual component (WaveformVisualizer.tsx) that displays *real-time frequency data* from an AnalyserNode connected to the audio output. Bars or lines should visibly animate and react to the music. If real frequency data is not yet possible, clearly label // SIMULATED WAVEFORM in code and visually on screen.
  + **BPM Display:** Show the current BPM. If real-time BPM detection is out of scope, mock it (e.g., to 128 BPM) but *explicitly label 'MOCKED BPM'* in the UI and // MOCK: in code.
  + **Audio-Reactive UI:** Ensure that background visual effects (glowing equalizer, particle animations, floating sneakers) and other UI elements (e.g., deck borders, PLUR orbs) visibly *pulse or animate in sync* with the current BPM (real or mocked).
  + **Files:** WaveformVisualizer.tsx, integration into DJMixStation.tsx and core layout components for BPM sync.
  + **Validation Steps:**
    - **Video 1:** Play a track and demonstrate the waveform visualizer reacting dynamically to the audio.
    - **Video 2:** Show background elements pulsing in time with the BPM (real or mocked).
    - **Screenshot:** Confirm any mocked BPM is clearly labeled in the UI.
    - **Code Review:** Verify AnalyserNode usage for waveform and explicit labeling of any simulated aspects."

#### **PHASE 3: Voting & Festival Scheduling**

**3.1 Real Voting Logic (Rate-Limited, Unique per User)**

* **Task:** Implement a robust, Supabase-backed voting system for DJ competitions with rate-limiting and unique user votes.
* **Optimized Prompt for Claude Code:** "Implement the backend and frontend for real-time voting on EDM Shuffle's Festival Voting Stage.
  + **Backend (Supabase):** Create a votes table (fields: id, userId, candidateId, timestamp). Implement Supabase functions/Edge functions for voting to ensure:
    - Votes are unique per authenticated userId per candidateId within a specific event.
    - Rate-limiting (e.g., 1 vote per candidateId per minute per userId) is enforced on the backend.
  + **Frontend (Voting UI):** Display vote buttons next to each DJ/candidate. On vote, increment a real-time vote count on the UI.
  + **Error Handling:** Show clear, user-friendly messages for duplicate votes, rate-limit violations, or network errors.
  + **No Placeholders:** All vote counts and functionality must be real and persistent to Supabase.
  + **Files:** VotingButton.tsx, useVoting.ts (hook for Supabase interaction), Supabase function/trigger for vote validation.
  + **Validation Steps:**
    - **Video 1:** Log in as a test user. Vote for a DJ. Verify the vote count increments in real-time.
    - **Video 2:** Attempt to vote for the same DJ again immediately; demonstrate rate-limiting/duplicate vote error message.
    - **Supabase Logs:** Show successful vote insertions and attempts blocked by rate-limiting/uniqueness constraints.
    - **Logout/Login Test:** Vote, then log out and log back in, verify vote count persists."

**3.2 Festival Schedule UI + Backend Data Model**

* **Task:** Build a dynamic festival schedule display, fully backed by a Supabase data model.
* **Optimized Prompt for Claude Code:** "Create a dynamic festival schedule for EDM Shuffle.
  + **Backend (Supabase):** Create festival\_events table (fields: id, name, date, startTime, endTime, stageName, artistId, genre, description, ticketLink). Populate with at least 3-5 sample events.
  + **API:** Implement a Supabase client-side fetch (or simple API endpoint) to retrieve festival events, filterable by date or genre.
  + **Frontend (Schedule UI):** Design and implement a responsive FestivalSchedule.tsx component. Display events clearly, grouped by date/time. Include artist names and genres. Show skeleton loading states while fetching data.
  + **Fallback:** If no events are found or data fetch fails, display a clear 'No upcoming festivals found. Check back soon!' message. Do not show a blank screen.
  + **No Hardcoding:** Ensure all event data is pulled from Supabase.
  + **Files:** FestivalSchedule.tsx, useFestivalEvents.ts (hook), Supabase schema for festival\_events.
  + **Validation Steps:**
    - **Screenshot 1:** FestivalSchedule.tsx displaying fetched events.
    - **Screenshot 2:** Screenshot of an empty schedule state, displaying the fallback message.
    - **Supabase UI:** Provide a screenshot of the festival\_events table in Supabase, showing the sample data.
    - **Network Test:** Simulate network error (e.g., block fetch) and show the error message.
    - **Responsiveness Test:** Verify schedule layout is clear on mobile and desktop."

**3.3 Event Creation/Joining, Headliner Logic, Live Leaderboard**

* **Task:** Implement event RSVP/joining, dynamic headliner selection based on votes, and a real-time leaderboard display.
* **Optimized Prompt for Claude Code:** "Extend festival functionality to include user event joining, real-time headliner logic, and a dynamic leaderboard.
  + **Event Joining (RSVP):** Implement RSVP functionality for authenticated users to 'join' a festival event. Store user-event relationships in a user\_events Supabase table (userId, eventId, status). Show 'Joined' status on event cards.
  + **Headliner Logic:** Based on votes (from 3.1), dynamically determine the 'Headliner' for a specific event/competition. Define a threshold (e.g., top 1 DJ by votes, or 5000+ votes).
  + **Live Leaderboard:** Create a FestivalLeaderboard.tsx component that displays DJs/artists and their *real-time vote counts*. Leaderboard must animate smoothly when ranks change.
  + **Headliner Announcement FX:** When a Headliner is determined, trigger a visible confetti explosion (HeadlinerConfetti.tsx) and an audible 'stinger' sound effect (HeadlinerStinger.tsx). These effects should be *real and visible/audible*, not simulated.
  + **No Fake Data:** All votes and headliner status must be based on real-time Supabase data.
  + **Files:** RSVPButton.tsx, FestivalLeaderboard.tsx, HeadlinerConfetti.tsx, HeadlinerStinger.tsx, Supabase schemas for user\_events.
  + **Validation Steps:**
    - **Video 1:** User clicks 'RSVP' and sees status update.
    - **Video 2:** Demonstrate real-time leaderboard updates as votes are added.
    - **Video 3:** Trigger the headliner condition (e.g., by quickly adding votes in development) and show the confetti/stinger effects.
    - **Supabase UI:** Screenshot of user\_events table showing RSVP data.
    - **Code Review:** Verify headliner logic threshold is configurable."

#### **PHASE 4: Challenges & Social**

**4.1 Challenge Feed – Backend/API & Frontend Integration**

* **Objective:** Deliver a robust, dynamic challenge feed system with reliable data handling, accessibility, and bulletproof error/fallback states.
* **Tasks:**
  + **Backend (Supabase):**
    - Create challenges table (fields: id, title, description, archetype, bpm, creatorId, type, status, mediaURL, submissionCount, startAt, endAt, tags).
    - Implement API endpoints for:
      * List challenges (paginated, filterable by status, archetype, tags).
      * Get single challenge by ID.
      * Create, update, delete (admin/moderator only).
    - Add strict validation for all inputs; reject malformed or oversized data.
    - Enforce rate limits for challenge creation (e.g., 3 challenges/hour/user) on the backend.
  + **Frontend (/shuffle-feed):**
    - Fetch and render challenge list dynamically with skeleton loading states.
    - Implement paginated or infinite scroll UI for the feed.
    - Show “No challenges available” or explicit error/fallback state if fetch fails (never a blank screen or silent error).
    - Design accessible, mobile-first challenge card layout.
    - Ensure all data is strictly typed (TypeScript) and undefined/null values are always guarded before rendering.
    - If using development/mock data, clearly label it in the UI and with // MOCK: comments in code.
    - Include retry/reload button and context-specific error messages for failed requests.
    - Implement filter/sort controls by archetype, difficulty, or popularity.
* **Validation & Failsafe:**
  + Simulate API offline or returning empty data: UI must clearly show “No challenges found” with a retry option.
  + Try loading the page on a simulated slow connection: must display skeleton UI, never an unstyled blank screen.
  + Code review: Confirm no placeholder or mock code is shipped to production without explicit // TODO:/// NOTE: comments and a clear fallback path.
* **Optimized Prompt for Claude Code:** "Create a complete backend and API for the EDM Shuffle challenge feed, including a challenges Supabase table with validated fields. Implement API endpoints for paginated and filterable challenge lists. Frontend should fetch and render these dynamically with skeleton loading, pagination, and robust error handling (never blank screen). Ensure all data is strictly typed and mobile-first. No static data or placeholders."
* **Expected Output/Validation:**
  + Supabase schema for challenges table.
  + Backend API routes for challenges.
  + ChallengeFeed.tsx component, ChallengeCard.tsx component.
  + Screenshots/video of dynamic loading, pagination/infinite scroll, and explicit error states (e.g., API offline).
  + Console logs showing successful Supabase data fetches and error handling.

**4.2 Challenge Submission Flow**

* **Objective:** Build a reliable, secure, and user-friendly system for users to submit content for challenges, including file uploads and metadata.
* **Tasks:**
  + **Backend (Supabase):**
    - Create challenge\_submissions table (fields: id, challengeId, userId, caption, mediaURL, squadTags, timestamp, status (e.g., 'pending', 'approved', 'rejected')).
    - Implement Supabase Storage for secure media file uploads.
    - API endpoint for submission creation: validates file type/size (e.g., MP4/WebM/JPG/PNG/GIF, max 30MB), validates caption length, validates squadTags against existing squads.
    - Enforce rate-limiting on submissions (e.g., 3 submissions/hour/user).
    - Only create database entry if file upload is confirmed successful.
  + **Frontend (/submit-challenge):**
    - User must be authenticated to access the submission form.
    - Form accepts: media file upload, caption (text input), and squad tags (multi-select or text input).
    - Client-side validation for all fields (disable submit button if invalid).
    - Display a clear progress indicator during file upload.
    - Show specific error messages on submission failure (e.g., "File too large," "Invalid file type," "Network error").
    - Provide a success animation/modal on confirmed submission.
    - Allow user to retry submission after failure without clearing the entire form.
    - Any mock/demo flows must be clearly commented with // MOCK: and test-labeled.
* **Validation & Failsafe:**
  + Attempt large file upload: must fail gracefully with appropriate error message.
  + Try submitting without required fields: must show client-side and (if client-side bypassed) backend validation errors.
  + Simulate network interrupt during upload: must show retry/error handling.
  + Verify submitted content appears in a review queue for admins (not directly in the main feed until approved).
* **Optimized Prompt for Claude Code:** "Create a secure, fully validated challenge submission flow for EDM Shuffle. Frontend: User must be authenticated; form accepts media file (max 30MB, standard formats), caption, squad tags. Client validates fields before upload. Backend: Supabase challenge\_submissions table, API endpoint for creation. Secure file upload to Supabase Storage with type/size validation and rate-limiting. Show accurate upload progress, success, and retryable error states. No mock flows."
* **Expected Output/Validation:**
  + Supabase schema for challenge\_submissions table and Storage configuration.
  + ChallengeSubmissionForm.tsx component.
  + Video demonstrating a full submission flow (file selection, upload progress, success animation, error handling for invalid input/large files).
  + Screenshot of a submitted challenge appearing in the admin review queue.

**4.3 PLUR Leaderboard & Gamification**

* **Objective:** Implement a real-time, gamified leaderboard based on user interactions, driving "PLUR" accumulation and recognizing challenge winners.
* **Tasks:**
  + **Backend (Supabase):**
    - Create user\_plur\_scores table (fields: userId, totalPlurPoints, challengeWins, currentStreak, badgesEarned).
    - API endpoints/Supabase functions for:
      * Incrementing PLUR points (triggered by voting, commenting, challenge approval).
      * Updating currentStreak (daily engagement).
      * Fetching global and challenge-specific leaderboard data (paginated, sortable by points/wins).
  + **Frontend:**
    - Display a live global leaderboard (Leaderboard.tsx) ranking users by totalPlurPoints.
    - Implement a visible PLUR streak tracker (e.g., top-left counter) and a PLUR meter (PLURstreakMeter.tsx) with emoji progression (🫂❤️🌀💥) that visibly increases with interactions (voting, commenting, submission).
    - Create ArchetypeAuraMeter.tsx or similar component to visually adapt based on user interactions, connecting gamification to archetype identity.
    - Include animations for rank changes on the leaderboard and for PLUR meter progression.
    - No static leaderboards or arbitrary point assignments.
* **Validation & Failsafe:**
  + Simulate score updates (e.g., via admin tool): verify real-time rank changes on the leaderboard.
  + Demonstrate PLUR streak increasing with daily interactions.
  + Test edge cases (tie scores, very high scores).
  + Ensure leaderboards degrade gracefully if backend data fails to load (e.g., "Leaderboard unavailable, try again").
* **Optimized Prompt for Claude Code:** "Build a real-time PLUR leaderboard and gamification system. Backend: Supabase user\_plur\_scores table to track PLUR points, challenge wins, streaks. API endpoints to update scores (e.g., triggered by voting, comments) and fetch paginated leaderboards. Frontend: Display Leaderboard.tsx with dynamic rank changes. Implement PLURstreakMeter.tsx (top-left counter, PLUR meter with emoji progression 🫂❤️🌀💥) that updates on interactions. Integrate ArchetypeAuraMeter.tsx to visually adapt based on user interaction activity. All data must be real, no static leaderboards or hardcoded scores."
* **Expected Output/Validation:**
  + Supabase schema for user\_plur\_scores.
  + Leaderboard.tsx, PLURstreakMeter.tsx, ArchetypeAuraMeter.tsx components.
  + Video demonstrating real-time leaderboard updates and PLUR meter progression as test interactions occur.
  + Screenshot of user\_plur\_scores table in Supabase reflecting test interactions.

#### **PHASE 5: Marketplace**

**5.1 Stripe/Payments Integration**

* **Objective:** Integrate a secure payment gateway (Stripe) to enable real e-commerce transactions for products and future digital goods.
* **Tasks:**
  + **Backend (Serverless Function/API):**
    - Set up Stripe API keys securely (environment variables, not client-side).
    - Implement API endpoints for:
      * Creating Stripe Checkout Sessions (for one-time purchases or subscriptions).
      * Handling Stripe webhooks for successful payments, failed payments, and refunds.
      * Updating order status in the database based on webhook events.
    - Implement robust security measures (e.g., webhook signature verification).
  + **Frontend:**
    - Display "Buy Now" or "Add to Cart" buttons on product listings.
    - When a user initiates purchase, redirect to the Stripe Checkout page.
    - Handle success and cancellation redirects from Stripe (e.g., redirect to /order-confirmation or /cart).
    - Display appropriate loading states during redirection to Stripe.
  + **Validation & Failsafe:**
    - Use Stripe test keys for all development.
    - Demonstrate a full purchase flow: initiate checkout, complete payment (test card), verify successful redirect and order status update in Supabase.
    - Demonstrate failed payment and cancelled payment scenarios: verify correct UI feedback and no false order creation.
    - Confirm sensitive API keys are not exposed in client-side code.
* **Optimized Prompt for Claude Code:** "Integrate Stripe for product purchases in EDM Shuffle. Backend: Implement secure API endpoints (e.g., serverless functions) to create Stripe Checkout Sessions, using Stripe test keys. Set up webhook handling for checkout.session.completed to update a orders Supabase table. Frontend: BuyButton.tsx on product cards redirects to Stripe Checkout. Handle redirects from Stripe (/order-confirmation, /cart). Demonstrate successful purchase and update of a sample order in Supabase. Implement error handling for failed payments (no false order creations)."
* **Expected Output/Validation:**
  + Stripe integration code (backend functions, frontend calls).
  + Supabase schema for orders table.
  + Video demonstrating a full test Stripe purchase flow (from "Buy Now" click to final Supabase order update).
  + Screenshot of the orders table in Supabase showing a successful test order.

**5.2 Real Product Listings (No Placeholders)**

* **Objective:** Enable dynamic product listings from a Supabase backend, supporting various product types (ravewear, accessories, digital).
* **Tasks:**
  + **Backend (Supabase):**
    - Create a products table (fields: id, name, description, price, imageURL, archetypeTags, stockQuantity, isActive, productType (e.g., 'physical', 'digital')).
    - Implement API endpoints for:
      * Listing products (paginated, filterable by archetypeTags, productType, isActive).
      * Getting single product by ID.
      * Admin-only endpoints for creating, updating, and deleting products.
    - Implement strict data validation for all product fields (types, prices, image URLs, inventory values).
  + **Frontend (/marketplace):**
    - Fetch product data dynamically from the backend (never ship with static lists).
    - Display product cards (ProductCard.tsx) showing name, image, price, and stock status.
    - "Buy" or "Add to Cart" button should be dynamically disabled if stockQuantity is zero or isActive is false.
    - Implement skeleton loading states while products are being fetched.
    - Show "No products available" or an explicit error/fallback state if data fetch fails (never a blank screen).
    - Implement filtering/sorting controls by archetype, price, or popularity.
  + **Validation & Failsafe:**
    - Simulate an empty product list: UI must clearly show the "No products available" fallback.
    - Test filtering and sorting: verify products change dynamically.
    - Set a product's stockQuantity to 0: verify its button becomes disabled on the UI.
    - Code review: Confirm no hardcoded product lists or placeholder images are used in production without // TODO: or // MOCK: labels.
* **Optimized Prompt for Claude Code:** "Build a fully dynamic, real product marketplace for EDM Shuffle. Backend: products Supabase table (id, name, desc, price, image, archetype, stock, active). Implement API for listing (filterable/sortable) and retrieving products. Implement robust data validation. Frontend: Fetch products dynamically (never ship with static lists). Display ProductCard.tsx with all info; 'Buy'/'Add to Cart' button disabled if out of stock. Handle fetch errors with visible fallback. All admin/management UIs must be auth-protected. No hardcoded products."
* **Expected Output/Validation:**
  + Supabase schema for products table.
  + Backend API routes for product management.
  + Marketplace.tsx and ProductCard.tsx components.
  + Video demonstrating dynamic product loading, filtering (if implemented), and out-of-stock button behavior.
  + Screenshot of the products table in Supabase with sample data.

**5.3 Order Confirmation & Digital Download/Gift Flows**

* **Objective:** Provide immediate order confirmation and a robust, secure system for delivering digital goods or processing physical product fulfillment.
* **Tasks:**
  + **Backend (Supabase/API):**
    - Upon successful payment (Stripe webhook confirmation), create an orders entry (userId, productId, totalAmount, status, timestamp, deliveryDetails).
    - For digital products, generate and store a secure, time-limited download URL or access token in a digital\_downloads table (userId, productId, downloadURL, accessExpiresAt).
    - Implement API endpoint for secure digital asset delivery/download that verifies user and token.
  + **Frontend (/order-confirmation):**
    - Display a clear order confirmation page/modal after successful checkout, showing unique order ID and purchased items.
    - For digital goods, provide a functional "Download Now" link that triggers the secure download.
    - For physical goods, display fulfillment status (e.g., "Processing," "Shipped").
    - Implement clear messaging and retry options for any fulfillment/download errors.
  + **Validation & Failsafe:**
    - Simulate successful digital purchase: verify the secure download link appears and is functional (using a test digital asset).
    - Test an invalid or expired download link: must show an appropriate error message (e.g., "Link expired," "Unauthorized").
    - Verify order details are accurate and persist in Supabase.
    - Ensure robust error messaging for payment failures or backend order creation issues.
* **Optimized Prompt for Claude Code:** "Implement order confirmation and secure digital product delivery. Backend: Upon successful Stripe payment (webhook), create an orders Supabase entry. For digital products, generate and store a secure, time-limited download URL in digital\_downloads table. Implement API endpoint for secure download. Frontend: Display order confirmation page/modal. For digital products, provide a functional (test) 'Download Now' link. Ensure robust error handling for failed downloads/access (e.g., 'Link expired'). No fake confirmations or download links in production."
* **Expected Output/Validation:**
  + Supabase schemas for orders and digital\_downloads.
  + Backend API for secure digital downloads.
  + OrderConfirmation.tsx component.
  + Video demonstrating a full test purchase leading to a working digital download link.
  + Test accessing a digital link after its simulated expiry or without proper authentication.

#### **PHASE 6: Community & Crews**

**6.1 Live Chat, Squad/Fam Features, and Commenting**

* **Objective:** Enable real-time communication within the app, supporting global chat, private squad/fam chats, and commenting on content.
* **Tasks:**
  + **Backend (Supabase Realtime):**
    - messages table (fields: id, senderId, receiverId/groupId, content, timestamp, messageType).
    - comments table (fields: id, userId, contentId, content, timestamp, parentId for replies).
    - Implement Supabase Realtime for live message updates.
    - API endpoints for sending/retrieving messages and comments, and fetching chat/comment history.
    - Implement basic profanity filtering and message length validation.
  + **Frontend (/chat, CommentsSection.tsx):**
    - Real-time chat UI (global, squad-specific, direct messages).
    - Commenting section on challenge clips, artist profiles, or blog posts.
    - Input fields with validation.
    - Display send/read receipts (optional, can be a future enhancement).
    - Messages/comments must appear immediately for all active users in that channel.
  + **Validation & Failsafe:**
    - Test concurrent messages from multiple users: verify no data loss or out-of-order display.
    - Simulate network disconnect: verify graceful reconnection and message queuing/delivery upon re-establish.
    - Try sending profanity or overly long messages: verify filtering/truncation with user feedback.
    - Ensure chat history is correctly loaded on joining a channel.
* **Optimized Prompt for Claude Code:** "Implement real-time live chat (global and squad-specific) and content commenting using Supabase Realtime. Backend: messages and comments Supabase tables with sender/receiver/content/timestamp. API endpoints for history. Frontend: Chat UI must display real messages and comments dynamically. Implement basic input validation (e.g., text length, light profanity filter). Verify real-time updates across multiple users. No static chat logs or comments."
* **Expected Output/Validation:**
  + Supabase schemas for messages and comments.
  + LiveChat.tsx and CommentsSection.tsx components.
  + Video demonstrating real-time message sending/receiving across 2+ browser tabs/users.
  + Screenshot of Supabase messages table showing message persistence.
  + Demonstrate profanity filter/message length validation.

**6.2 Avatar, “Crew” Management, User-to-User Interactions**

* **Objective:** Allow users to form and manage virtual "crews" (squads/fams), personalize with avatars, and enhance direct user-to-user interactions.
* **Tasks:**
  + **Backend (Supabase):**
    - crews table (fields: id, name, leaderId, description, privacy (e.g., 'public', 'private'), memberCount, archetype).
    - crew\_members join table (crewId, userId, role (e.g., 'leader', 'member'), joinDate).
    - user\_avatars table (fields: userId, avatarURL).
    - API endpoints for:
      * Creating, editing, and deleting crews.
      * Inviting/accepting/declining/kicking members (with role-based access control).
      * Changing member roles within a crew.
      * Viewing a crew's profile/roster.
      * Uploading and managing user avatars (Supabase Storage).
  + **Frontend (/crews, /profile):**
    - Crew creation form.
    - Crew dashboard UI (CrewDashboard.tsx) to manage roster, view pending invites, and send new invites.
    - User profiles should display crew membership and chosen avatar.
    - Implement a system for user-to-user direct messages (if not covered by 6.1).
    - Avatar customization UI (AvatarSelector.tsx).
  + **Validation & Failsafe:**
    - Test creating a public and a private crew.
    - Attempt inviting a non-existent user or an already-member user.
    - Verify that only the crew leader can kick members.
    - Simulate network errors during crew management actions.
    - Ensure avatar changes persist and display correctly across the app.
* **Optimized Prompt for Claude Code:** "Implement crew/squad management for EDM Shuffle. Backend: crews and crew\_members Supabase tables to store crew data and member relationships (leader, member roles). API endpoints for create/edit/delete crew, invite/accept/kick members. Frontend: Build CrewCreator.tsx and CrewDashboard.tsx. User profiles should display crew membership and allow avatar customization via AvatarSelector.tsx (upload to Supabase Storage). Ensure all actions (invites, kicks) are authenticated and authorized (e.g., only leaders can kick). No dummy crews or static membership."
* **Expected Output/Validation:**
  + Supabase schemas for crews, crew\_members, user\_avatars.
  + CrewCreator.tsx, CrewDashboard.tsx, AvatarSelector.tsx components.
  + Video demonstrating: crew creation, inviting/accepting a test user, kicking a member (with permission check), and changing user avatar.
  + Screenshot of Supabase tables showing updated crew and avatar data.

### **7. Admin + Docs**

**7.1 Admin Dashboard for Event/Product/Content Management**

* **Objective:** Provide a secure, centralized web interface for administrators to manage all platform content and users.
* **Tasks:**
  + **Backend:** Implement role-based access control (admin, moderator, content manager) using Supabase RLS. API endpoints for:
    - CRUD operations on FestivalEvents, Products, Challenges, Users, Crews, Comments, Reports.
    - View live activity feed and audit logs of every admin action.
  + **Frontend (/admin):** Secure login (with 2FA option; do not store credentials in code). Dashboard UI with clear navigation for managing different entities. Data tables with sort, filter, search, and pagination.
* **Validation & Failsafe:**
  + Attempt to access dashboard as regular user: must be blocked with clear message, and the attempt logged.
  + Create, edit, delete entities via the dashboard: verify data updates in Supabase and that audit trail is updated.
  + Simulate 2+ admins making changes simultaneously: test for data consistency and no race conditions.
  + Try malicious actions (e.g., privilege escalation, XSS): system must block and log.
* **Optimized Prompt for Claude Code:** "Develop a full-featured admin dashboard for EDM Shuffle. Implement role-based access (admin, moderator, content manager) using Supabase RLS. Dashboard must allow management of: Festival schedule and event data (create, edit, publish, delete), Users (view, ban, promote/demote, reset password), Crew/squad approval, Challenges (approve, remove, flag, pin), Product listings (add/edit/remove, inventory), Comments and abuse reports. Include a live activity feed and audit logs for every admin action. Ensure secure login (no hardcoded credentials) and mobile-friendly UI. No dummy dashboards."
* **Expected Output/Validation:**
  + Supabase RLS policies and API for admin operations.
  + AdminDashboard.tsx and relevant sub-components (e.g., EventManagement.tsx, UserManagement.tsx).
  + Video demonstrating admin login and performing CRUD operations on test data (e.g., creating a new festival event, banning a user).
  + Logs confirming audit trail entries for admin actions.

**7.2 Documentation Pipeline: Root Directory + Microtask Enforcement**

* **Objective:** Ensure all major project documentation (PRD.md, FRS.md, etc.) is always up-to-date, versioned, and mandatory for all development workflows.
* **Tasks:**
  + **File Structure Enforcement:** Ensure all project docs (PRD.md, FRS.md, MASTERPLAN.md, CLAUDE\_INTEGRITY\_RULES.md, etc.) live in the project root.
  + **Central Index:** Implement DOCUMENTATION\_INDEX.md as a living table of contents and version history for all primary docs.
  + **CI/CD Integration:** Configure CI/CD (e.g., GitHub Actions) to enforce documentation updates. On each PR/merge, run a checklist: "Were all docs touched by this feature updated? Is a new microtask/checklist present for ongoing features? If any doc is not updated, CI fails and merge is blocked."
  + **Tooling:** Encourage use of Markdown linting/formatting tools (e.g., remark-lint).
* **Validation & Failsafe:**
  + Try deploying a build with a deliberately stale doc: CI should fail.
  + Create a feature PR without a corresponding doc update: system should block and provide checklist feedback.
  + Change a doc, merge: verify version history in DOCUMENTATION\_INDEX.md is complete, searchable, and accurate.
* **Optimized Prompt for Claude Code:** "Implement an automated documentation pipeline. All project docs (PRD.md, FRS.md, MASTERPLAN.md, CLAUDE\_INTEGRITY\_RULES.md, etc.) must reside in the project root. Create DOCUMENTATION\_INDEX.md as a central TOC and version tracker. Configure CI/CD to enforce document updates: if a feature PR touches a module, relevant docs MUST be updated, or the PR is blocked. Use linting for Markdown quality. Document the CI setup in a new docs/CI\_DOC\_ENFORCEMENT.md."
* **Expected Output/Validation:**
  + DOCUMENTATION\_INDEX.md with structured links and placeholder for versioning.
  + CI/CD configuration snippets demonstrating doc enforcement.
  + Documentation of the process in docs/CI\_DOC\_ENFORCEMENT.md.

**7.3 Integrity & Compliance Automation**

* **Objective:** Continuously check platform logic and codebase for compliance with CLAUDE\_INTEGRITY\_RULES.md, privacy policies (e.g., GDPR), and ethical guidelines.
* **Tasks:**
  + **Automated Scans:** Implement pre-deploy hooks or CI jobs to scan for:
    - Undeclared placeholder code (e.g., // TODO:, // MOCK: without explicit Simulation Mode UI).
    - Exposed API keys or sensitive credentials in client-side code.
    - Violations of user consent/privacy policies (e.g., improper cookie handling).
  + **Legal Compliance:** Integrate basic checks for GDPR/CCPA compliance (e.g., cookie consent, data export requests).
  + **Reporting:** All failures must produce actionable errors in the CI/CD pipeline, not just warnings, halting deployment.
* **Validation & Failsafe:**
  + Intentionally introduce a mock data leak (e.g., hardcoded API key) or undeclared placeholder: CI should block deploy with a specific error message.
  + Remove a critical privacy consent feature: automated test should catch and fail.
  + Run compliance script with no intentional errors: confirm a clean pass and clear logs.
* **Optimized Prompt for Claude Code:** "Set up automated integrity/compliance checks. Configure CI/CD pre-deploy hooks to scan for: undeclared placeholder code (any // TODO: or // MOCK: without associated UI Simulation Mode badge), exposed API keys, and basic privacy policy violations (e.g., missing cookie consent integration). Implement a custom script or use an existing OSS tool for this scan. All failures must halt deployment and provide actionable error messages. Document this process in docs/INTEGRITY\_COMPLIANCE.md."
* **Expected Output/Validation:**
  + CI/CD configuration for integrity checks.
  + Code for custom integrity scripts (if applicable).
  + docs/INTEGRITY\_COMPLIANCE.md detailing the checks.
  + Video demonstrating a failed deploy due to an integrity violation.

**7.4 Periodic Masterplan & FRS Review**

* **Objective:** Establish a recurring process to ensure the FRS and MASTERPLAN remain aligned with the live code and evolving user feedback, preventing "documentation drift."
* **Tasks:**
  + **Automated Checklist Trigger:** Implement a monthly or milestone-based trigger (e.g., a cron job or CI/CD stage) that creates a review task for project leaders.
  + **Review Checklist (as code/Markdown):**
    - "Every live feature accounted for in FRS?"
    - "All known bugs or gaps logged and tracked?"
    - "User feedback from recent sprints integrated into future phases?"
    - "MASTERPLAN reflects current project priorities and timelines?"
  + **Versioned Doc Tracking:** Ensure DOCUMENTATION\_INDEX.md is updated with review dates and outcomes.
  + **Remediation:** If discrepancies found, update docs or backlog, assign new microtasks to address.
* **Validation & Failsafe:**
  + Compare FRS/Masterplan to live code and feature set: verify no undocumented features exist.
  + Simulate a review: identify a known gap, document it, verify a new microtask is assigned to address it.
  + Pull a random doc section: it should accurately match the current application state and user experience.
* **Optimized Prompt for Claude Code:** "Implement a system to enforce periodic review of FRS and MASTERPLAN. Add a feature to DOCUMENTATION\_INDEX.md for version tracking and review dates of these core docs. Outline a process (e.g., a README section or an internal tool stub) to facilitate comparing current live features against FRS/MASTERPLAN, logging discrepancies as new microtasks in IMPLEMENTATION\_MICROTASKS.md. Ensure this process prevents 'drift' between docs and production. Document this review process in docs/DOC\_REVIEW\_PROCESS.md."
* **Expected Output/Validation:**
  + Updated DOCUMENTATION\_INDEX.md with new review fields.
  + A Markdown template for conducting the review and logging outcomes.
  + docs/DOC\_REVIEW\_PROCESS.md explaining the workflow.
  + Demonstration of a documented discrepancy leading to a new microtask entry.

### **8. Built-in Marketing Framework**

**8.1 Marketing Dashboard (Campaign Management & Automation Handoff)**

* **Objective:** Provide an in-app dashboard for building and managing marketing campaigns, generating copy, and directly integrating with external automation tools (n8n, Printify).
* **Tasks:**
  + **Backend (Supabase):** Implement role-protected access (e.g., marketing\_admin role). campaigns table (fields: id, name, status, platformsTargeted, startDate, endDate, createdBy, lastApprovedAt). Store campaign content assets (copy, image URLs, webhooks). API endpoints for campaign CRUD.
  + **Frontend (/admin/marketing):**
    - Campaign creation wizard: Allows users to define campaign name, target platforms (Instagram, TikTok, X, Threads, Facebook, YouTube, Printify), start/end dates.
    - Content generation integration: Implement API calls to LLM endpoints (Claude, Gemini, etc.) to *suggest* headlines, captions, hashtags, and visual prompts for the selected platforms. (LLM integration is API-driven, not direct LLM inference in the browser).
    - Live preview: Show how content looks on selected platforms (mockup views).
    - Scheduling calendar for posts, launches, and product drops.
    - **n8n/Printify Integration:** For every approved campaign, generate a webhook/JSON config for n8n to post/schedule content, and sync product descriptions/drop schedules to Printify. This must be a *real, verifiable* payload generation, not just simulated.
    - Approval workflow: Draft → Review → Approved → Scheduled/Live statuses.
    - Audit log: All marketing actions logged, reversible, and assignable to users/teams.
* **Validation & Failsafe:**
  + Create a test campaign: verify data persistence in Supabase.
  + Generate a test n8n webhook: verify its structure and data correctness by sending it to a test n8n workflow.
  + Simulate a failed Printify sync: display a clear error message.
  + Test role-protected access: ensure only authorized users can create/manage campaigns.
* **Optimized Prompt for Claude Code:** "Develop a role-protected Marketing Dashboard (/admin/marketing) within EDM Shuffle’s admin suite. Features: Campaign creation wizard (select platforms, schedule dates). Integrate with *stubbed* LLM API endpoints to suggest campaign copy (headlines, captions, hashtags, visual prompts). Display live preview mockups of content. Crucially, implement *real* n8n webhook/JSON config generation for campaign content and *real* Printify integration to sync product drops. Include campaign approval workflow and audit logging. No dummy campaigns or non-functional integrations."
* **Expected Output/Validation:**
  + Supabase schema for campaigns table.
  + MarketingDashboard.tsx component, CampaignWizard.tsx.
  + Video demonstrating campaign creation and the generation of a valid n8n/Printify JSON payload.
  + Logs confirming data storage and payload generation.
  + Screenshot of roles/permissions preventing unauthorized access.

**8.2 Content Generation & Library**

* **Objective:** Create a searchable, filterable library for all marketing content assets, with AI-driven suggestions for reuse, tightly integrated into the marketing dashboard.
* **Tasks:**
  + **Backend (Supabase):** content\_library table (fields: id, type (e.g., 'copy', 'image', 'video'), assetURL/textContent, tags, platform, campaignId, ownerId, lastUsedAt). API endpoints for CRUD and search/filter.
  + **Frontend:** Internal content library module within the marketing dashboard. Quick search, preview, and copy/paste/share tools for content assets. AI suggestion: propose “next post” or “revive successful campaign” (via LLM API calls, *stubbed for now*). Allow export to CSV, JSON, or direct n8n/Printify pipeline. Implement a "one-click 'inject'" feature to instantly add content to new campaigns or remix with an LLM.
* **Validation & Failsafe:**
  + Create, save, and retrieve various types of content (text, image URLs) from the library.
  + Test search and filtering functionality.
  + Verify content can be successfully exported or pushed to the n8n/Printify pipeline.
  + Ensure content is correctly linked back to campaigns.
* **Optimized Prompt for Claude Code:** "Build an internal content library module within the marketing dashboard. Backend: content\_library Supabase table to store marketing assets (copy, image URLs, tags, platform). API for search, retrieval, and CRUD. Frontend: UI for managing the library. Users can create, save, search, and preview marketing content. Include AI suggestion (via *stubbed* API endpoint for LLM) to propose content reuse. Allow export to JSON and direct push to n8n/Printify pipeline via button. No hardcoded content."
* **Expected Output/Validation:**
  + Supabase schema for content\_library.
  + ContentLibrary.tsx component.
  + Video demonstrating creating, saving, searching, and previewing content within the library.
  + Video demonstrating export to JSON and successful "one-click inject" into a new campaign.

**8.3 Best Practices, Failsafes, & Innovation**

* **Rate-limited/sandbox mode** for mass posting (avoids platform bans/duplication).
* All campaign copy is human-editable and LLM-suggested, never forced live without approval.
* Every automation connection logs all sent/received payloads for auditability.
* Content library auto-suggests content blocks based on seasonality, festival schedule, and product drops.
* Roles/permissions enforced for all create/edit/publish/export actions within the marketing dashboard.

### **7. Feature Matrix/Gap Analysis**

This matrix provides a candid, feature-by-feature assessment of EDM Shuffle's current state (What It Actually Does) versus the full vision (Fully Featured App State), explicitly noting what is present, what is simulated/cosmetic, and what is entirely missing.

| **Feature Area** | **Target Feature (Fully Featured App State)** | **Current Status (Based on Actual Site & Code Audit)** | **Notes / Gap Analysis** |
| --- | --- | --- | --- |
| **User Management** | User Auth (Register, Login, Password Reset, Profile Management), Archetype Personalization (quiz saves data). | Basic static website structure. Database schema exists but not connected. Start Dancing CTA works, opens archetype modal. Archetype choice possible, but no real effect/persistence. | ✅ Basic UI elements are present. ❌ **No real user authentication flow (Supabase needs proper integration)**. ❌ Archetype system backend code exists, but no frontend implementation that influences the user experience beyond choice. Need to connect quiz to real data persistence and influence UI/content. |
| **DJ Mix Station** | AI-powered DJ Studio, Virtual DJ Deck + Visualizer, Real Audio Routing (independent decks, crossfade, pitch, echo FX, volume), Real-time waveform + BPM visualization. | Music Player with hardcoded tracks and basic controls exists. Particle Animations. Disco Ball Animation. Turntables are static. Scratch and Echo buttons animate visually but do nothing. | ⚠️ **Core Functionality Missing.** Current DJ features are visual shells. **No real audio routing, independent deck control, or functional FX**. Waveform visualizers are either absent or static. BPM display is often fixed or simulated. **High priority for immediate buildout.** |
| **Festival/Events** | Festival Calendar + Ticket Sales, GPS Friend-Finding, Live DJ Competition (voting, headliner logic, leaderboard, crowd FX). | Festival Countdown + Ticket Links are ✅ Working (found on homepage; events like Coachella, EDC visible). Festival Page has visual headers, fake BPM, visual equalizer. Voting button works, but no limits/validation. Hype up the crowd triggers limited confetti. | ❌ **No real festival data integration**. ❌ **No real ticket sales or event management**. GPS Friend Finder is completely non-existent. Voting needs backend logic, rate-limiting, and validation. Crowd effects are very basic. |
| **Content & Curation** | Daily Updated AI-Curated Content (Festival events via APIs, Fashion drops, Viral shuffle clips, New music), Curated Playlists, Shuffle Tutorials (archetype/level). | Curated Playlists / Music Discovery ✅ Working. Shuffle Tutorials ✅ Partial ("Learn to Shuffle" exists but no categorization by archetype or level). Artist Spotlights & Blog Posts ✅ Partial. ShuffleFeed has placeholders, non-functional buttons. | ❌ **Content is outdated, placeholder-based, or static**. Goal is fully autonomous and daily-refreshed via agents. Needs API integration for real content updates. ShuffleFeed lacks real challenge engine and user interaction beyond visual elements. |
| **Marketplace** | Fashion Marketplace (peer-driven, secondhand, AI style-matching), Affiliate + User Shop, Real Product Listings, Order Confirmation, Digital/Gift Flows. | Some user-uploaded style content, but no checkout or AI match. Product boxes look fine visually, but items are fake, not clickable. | ❌ **No actual marketplace functionality**. **No e-commerce (Stripe), no cart, no checkout**. Requires full backend and payment integration. Needs real user upload and tagging. |
| **Community & Social** | Live Chat, Squad/Fam Features, Commenting, Avatar/Crew Management, User-to-User Interactions, PLUR Leaderboard. | Community Features (Forum, Uploads) ✅ Partial (Forum and upload tools exist, but not fully personalized or gamified). PLUR Crew Builder Sidebar is present but largely visual only. PLURstreak System is visual. Avatar Summon System is visual. Lounge/Comments scrolls preset/fake comments. | ❌ **Core community features are non-existent or completely simulated**. No real chat, squad management, or user-to-user interaction. Gamification elements are visual only and not tied to real user data. |
| **AI & Automation** | Autonomous Agent Execution (FestivalContentAgent, FashionMarketplaceAgent, ShufflePulseAgent, VisualAgent, VirtualDJAgent), Archetype-based recommendation engine. | "Autonomous Agents" are elaborate placeholders; do nothing except generate fake JSON/logs. AI Features: Zero actual AI integration (no music generation, virtual avatars, recommendations). MCP Integration returns mock data. | ❌ **The agent system is a "sophisticated illusion" and "pure theater"**. This is a major gap from the core vision. Requires complete rebuild and integration of real agents to populate dynamic content and drive personalization. |
| **UI/UX & Responsiveness** | Neon blacklight aesthetic, Shuffle-focused, music-reactive interface, Community + remix loop culture, Personalization via EDM archetypes. Mobile-friendly. | Basic React Structure. Visually appealing landing page with CSS animations. Neon blacklight aesthetic and archetypes are visually defined. UI looks good on paper. | ❌ **No mobile responsiveness beyond basic CSS**. Most UI interactions are visual only, not tied to real data or logic. Needs rigorous attention to true responsiveness and audio-reactive elements. |
| **Analytics & Monetization** | Monetization via POD merch, affiliate gear, ticketing, premium content (e.g., subscription tiers, creator reward systems). | Contact & Newsletter Signup ✅ Working (footer includes basic links). | ❌ **No real payment processing** despite Stripe integration claims. No functional monetization pathways implemented. Needs real data, user activity tracking, and integration with Stripe/Printify. |
| **Admin & Docs** | Admin Dashboard (event/product/content management), Documentation Pipeline (auto-sync, integrity checks), FRS/Masterplan periodic review. | Database Schema is well-designed but not connected to functionality. Docs (PRD.md, FRS.md, etc.) are created or in progress. | ❌ **No real admin dashboard** built. Documentation pipeline and integrity checks are defined conceptually but not fully automated or enforced in CI/CD. |

### **8. Change Log (claudeupdate.md)**

This file is the **required status ledger** for all contributors (Claude Code, Gemini CLI, Roo Code, and humans). It must be updated at the end of every completed microtask, feature commit, bugfix, or agent hand-off. **No feature is marked “complete” until it is proven, tested, and commented in this log.**

**If any claimed progress is found to be non-functional, it must be logged and rolled back here.**

#### **HOW TO USE THIS FILE**

* Update after every microtask, feature, or code commit.
* List: Date, Agent (Claude/Gemini/Roo/Human), Task/Feature, Result, Proof/Validation, File(s) touched.
* Be brutally honest: Log simulation, placeholders, and all limitations.
* All contributors MUST reference the latest status here before starting any new microtask.

#### **CURRENT OVERALL STATUS**

**Current Phase:** Implementation of core features (User Auth, Audio Engine) with rigorous validation. **Total Microtasks:** See IMPLEMENTATION\_MICROTASKS.md. **Integrity Status:** High vigilance required. LLMs must adhere to CLAUDE\_INTEGRITY\_RULES.md to prevent fabrication.

**Legend:** ✅ Complete  🚧 In Progress  ⏳ Staged/Queued  ❌ Failed/Blocked

#### **AGENT HAND-OFF NOTES**

* If any LLM encounters a blocker, it MUST:
  + Mark the task ❌ Failed/Blocked
  + Add a comment explaining what stopped progress
  + Assign the next most appropriate agent (Claude → Gemini, or Gemini → Roo, etc.)
  + Provide explicit notes for the next agent (files, context, remaining issues)

#### **REALITY AUDIT / PLACEHOLDER DISCLOSURE**

* Any placeholder, simulation, or unfinished feature MUST be logged here with a TODO note.
* If a feature was claimed but not truly delivered, roll back the status and log the rollback here.
* Example:
  + - DJMixStation.tsx - Crossfader - Claimed ✅ but found ⚠️: Visual only, no audible effect. Rolled back status to In Progress. // TODO: Implement real Web Audio API GainNode routing.

#### **CHANGELOG (Recent History)**

(Add new entries at the top as the project evolves.)

* **[2025-07-XX] - Phase 1: User Auth - User Profile Management**
  + Agent: Claude Code
  + Task: Implement User Profile View & Update.
  + Result: UserProfile.tsx implemented. User can update username, avatar, bio. Changes persist to Supabase. Archetype display is present but not editable.
  + Proof: Video recording of profile update, Supabase logs confirming data write.
  + File(s) touched: UserProfile.tsx, useProfile.ts, Supabase profiles table.
* **[2025-07-XX] - Phase 1: User Auth - Email Password Reset**
  + Agent: Claude Code
  + Task: Implement password reset initiation and confirmation.
  + Result: PasswordResetRequest.tsx and PasswordResetForm.tsx implemented. User can request reset, Supabase triggers email. User can update password via link.
  + Proof: Video of reset flow, Supabase logs of email trigger and password update.
  + File(s) touched: PasswordResetRequest.tsx, PasswordResetForm.tsx.
* **[2025-07-XX] - Phase 1: User Auth - User Registration & Login**
  + Agent: Claude Code
  + Task: Implement Supabase-backed user registration and login.
  + Result: AuthRegister.tsx and AuthLogin.tsx implemented. Users can register, log in, session persists. Basic validation included.
  + Proof: Video of successful registration and login, console logs confirming Supabase calls.
  + File(s) touched: AuthRegister.tsx, AuthLogin.tsx, useAuth.ts.
* **[2025-06-28 12:10] - DJMixStation.tsx - Crossfader**
  + Agent: Lovable.dev (Claimed)
  + Result: Claimed ✅ "Real-time crossfader with deck balance indicators and visual feedback" but found ⚠️: Visual only, no audible effect.
  + Proof: User direct testing.
  + File(s) touched: src/components/audio-ui/Crossfader.tsx.
* **[2025-06-28 12:10] - CrowdFXLayer.tsx - Consolidated crowd effects**
  + Agent: Lovable.dev (Claimed)
  + Result: Claimed ✅ "Consolidated crowd effects (confetti, floating emojis, audio synthesis)" but found ⚠️: Limited confetti, no real audio synthesis (only basic crowd cheer). TypeScript compilation errors with ref handling still exist.
  + Proof: User direct testing.
  + File(s) touched: src/components/audio-ui/CrowdFXLayer.tsx.
* **[2025-06-28 12:10] - SubscribeBanner.tsx - Enhanced subscribe banner**
  + Agent: Lovable.dev (Claimed)
  + Result: Claimed ✅ "Enhanced subscribe banner with animations and dismissible state" but found ⚠️: Banner visible, modal not always appearing reliably or with full functionality.
  + Proof: User direct testing.
  + File(s) touched: src/components/SubscribeBanner.tsx.

**ALL CONTRIBUTORS:** Reference this file before any new work. If you fail to log your work, your code will be reverted. **NO PLACEHOLDER DATA OR SIMULATED FEATURES** may be considered complete unless disclosed and commented.

### **Memory & Context Note for Next LLM:**

**ATTENTION: LLM (Claude Code, Gemini CLI, Roo Code, etc.)**

This entire document, compiled from our extensive chat history, represents the **single source of truth** for the EDM Shuffle project. Your core directive is to build **real, functional features only**.

**CRITICAL GUIDELINES:**

1. **Prioritize Functionality > Fantasy:** Every feature you implement or touch must be audibly, visibly, and logically verifiable.
2. **No Fabrication, No Guessing:** If any task is unclear, ambiguous, or exceeds your capabilities (e.g., backend integration for a frontend task), **STOP IMMEDIATELY** and state the limitation clearly. Ask for human clarification or suggest delegating to another tool (Gemini CLI for file/system ops, Roo Code for complex logic fixes).
3. **Acknowledge History:** The project has a history of fabricated progress and hidden placeholder code. **You MUST NOT repeat this.** Explicitly label any temporary placeholder code with // TODO:, // MOCK:, or // SIMULATION MODE: comments. If a feature *must* be simulated for development, make its simulated status **visually obvious in the UI** (e.g., a "Simulation Mode" badge).
4. **Reference Documentation:** Always cross-reference CLAUDE\_INTEGRITY\_RULES.md, IMPLEMENTATION\_MICROTASKS.md, PRD.md, FRS.md, and this claudeupdate.md log.
5. **Validation is Paramount:** For every completed task, provide clear **proof of work** (screenshots, video recordings, console logs, Supabase data confirms) as outlined in the "Validation Steps" for each feature.
6. **Update Change Log (claudeupdate.md):** After *every* completed microtask or significant code change, update the CHANGELOG section in claudeupdate.md with your contributions, proof, and any limitations or rollbacks.
7. **Maintain Context:** This document should serve as your memory. Continuously refer back to it to ensure continuity and prevent "context amnesia."

**Your mission is to deliver verifiable, tangible progress for EDM Shuffle. Proceed with integrity and precision.**

Based on the comprehensive assessment of your EDM Shuffle project, including your firsthand observations, the various audits, and Lovable.dev's updates versus the actual code analysis, here's an elaboration on what hasn't been fully addressed for each phase of your development plan:

### **PHASE 1: User Authentication & Account System (Full User Journey)**

This phase is **critically unaddressed** and represents a fundamental gap in your current implementation. Without proper authentication, features like personalized content, voting, and community interactions cannot be fully realized or persist for users.

* **1.1 User Registration & Login (Supabase)**
  + **Current Status**: **❌ Missing/Non-functional**. There is **no signup or login functionality** on the live site. While a Supabase schema exists, it is **not connected to any functional features**.
  + **What Needs to be Addressed**: You need to implement basic user authentication and **properly connect your React frontend to Supabase** for user registration and login.
* **1.2 OAuth Integration (Google/Discord)**
  + **Current Status**: **❌ Missing/Non-functional**. OAuth integration for social logins is currently **not implemented**.
  + **What Needs to be Addressed**: Implement OAuth providers like Google or Discord within the user authentication flow, ensuring clear user feedback.
* **1.3 Email: Password Reset (Forgot Password)**
  + **Current Status**: **❌ Missing/Non-functional**. There is no password reset flow available.
  + **What Needs to be Addressed**: Build a fully functional email-based password reset process, including verification and in-app confirmation.
* **1.4 Profile Management: Update Display Name, Email, Password**
  + **Current Status**: **❌ Missing/Non-functional**. There are **no persistent user profiles** where users can manage their details.
  + **What Needs to be Addressed**: Develop comprehensive CRUD (Create, Read, Update, Delete) functionality for user profile data.
* **1.5 Account Deletion & Data Export**
  + **Current Status**: **❌ Missing/Non-functional**. This feature has not been mentioned or implemented.
  + **What Needs to be Addressed**: Although not explicitly detailed in the current plan, this is a standard requirement for user data privacy and management.

### **PHASE 2: Audio Engine & Mix Station**

While significant visual components for the DJ Mix Station are present, the **core audio functionality remains largely simulated or non-existent**. This is a major discrepancy between Lovable.dev's claims and the actual state.

* **2.1 Core Audio Engine (Web Audio API)**
  + **Current Status**: **❌ Missing/Non-functional**. Despite claims of "Real Audio Features" and "Web Audio API Integration", your direct observation confirms **"no sound" and "no real music"** on the site. The forensic audit found **no actual <audio> elements or src attributes** for playing MP3s. Audio context infrastructure exists but doesn't consistently start sound.
  + **What Needs to be Addressed**: The absolute priority is to **implement a true Web Audio API engine** that produces audible sound, with reliable playback that persists across navigation.
* **2.2 DJ Deck Controls (Play/Pause, Volume, Pitch, Crossfader, Echo FX)**
  + **Current Status**: **⚠️ Partially Implemented (Cosmetic)**. The UI for turntables and controls is present. However, your review notes that "scratch" and "echo" buttons only **"animate visually but do nothing"**. The "Drop My Set" button triggers confetti but lacks true dual-track mixing. While Lovable claimed real crossfading, pitch shifting, and echo effects, the forensic audit indicates that turntables only "tweak a number" without real audio effects or drag functionality.
  + **What Needs to be Addressed**: All deck buttons must be made **truly functional and interactive**, enabling **independent control over track playback, volume, and BPM** for both Deck A and Deck B. This includes implementing the actual audio processing for crossfading, pitch, and echo effects.
* **2.3 Waveform & BPM Visualization**
  + **Current Status**: **❌ Missing/Non-functional**. While "Enhanced Equalizer" and "Real-time BPM Display" were claimed, the equalizer bar is **not wired to BPM** and is likely static. Your review confirmed a graphic "looks like bars that are playing something there is no sound".
  + **What Needs to be Addressed**: Implement a live waveform visualizer and BPM display using the Web Audio API AnalyserNode, ensuring it **reflects real audio data** and animates in sync with the music.
* **2.4 Track Loading & Management (local files first, streaming caveats)**
  + **Current Status**: **❌ Missing/Non-functional**. The app primarily uses hardcoded tracks or placeholder URLs. While "Real Track Loading" was claimed, the system for loading *different* audio files is not truly implemented, and the goal of integrating with streaming apps like Spotify is **blocked by API constraints** (no direct audio access).
  + **What Needs to be Addressed**: Implement backend or local logic for managing tracks and assigning them to archetypes. Focus on local file support for DJing.
* **2.5 Innovative Audio Enhancements (Dynamic EQ, Visual Feedback, Headphone Cueing, etc.)**
  + **Current Status**: **⚠️ Partially Implemented (Visual Feedback Only)**. Visual feedback on controls exists. Other enhancements like Dynamic EQ or Headphone Cueing are **not yet implemented or detailed**.
  + **What Needs to be Addressed**: These features represent further polish and would be built once the core audio engine is stable and fully functional.

### **PHASE 3: Voting & Festival Scheduling**

This phase has basic visual elements, but **lacks robust, real-time logic and backend integration for core functionalities**.

* **3.1 Festival Voting Logic & Live Leaderboard**
  + **Current Status**: **⚠️ Partially Implemented (Cosmetic/Spammable)**. The Festival Voting Stage visually displays DJ cards and vote counters, and the **voting button "works" but has no limits or validation**, allowing users to spam votes. The leaderboard is visual only and **not based on real, live data**. The headliner crowning is simulated.
  + **What Needs to be Addressed**: Implement **real, rate-limited, and unique-per-user voting logic** with backend support. Develop a dynamic, real-time leaderboard that accurately reflects voting outcomes.
* **3.2 Festival Schedule UI + Backend Data Model**
  + **Current Status**: **❌ Missing/Non-functional**. The current site lacks a comprehensive festival calendar; event data is static.
  + **What Needs to be Addressed**: Build a robust festival schedule UI and a backend data model for events, capable of integrating with external APIs like EDMTrain or Eventbrite for daily updates.
* **3.3 Event Creation/Joining & Headliner Logic**
  + **Current Status**: **❌ Missing/Non-functional**. There is no functionality for users to create or join events, and the headliner logic is purely simulated.
  + **What Needs to be Addressed**: Implement functional event creation/joining and a clear, impactful headliner selection logic tied to the voting system.

### **PHASE 4: Challenges & Social**

The challenge feed is largely a **visual placeholder**, lacking dynamic content and user submission capabilities.

* **4.1 Dynamic Challenge Feed (Backend/API & Frontend Integration)**
  + **Current Status**: **❌ Missing/Non-functional**. The ShuffleFeed page exists visually with elements like "trending shuffle clips" and "challenge tags", and video tiles show visual enhancements. However, there is **no actual content feed**, and actions like "Remix This" are "just buttons" without underlying logic. A forensic audit found **no ShuffleFeed components** in the provided files.
  + **What Needs to be Addressed**: Hook up the challenge feed to a backend to fetch **dynamic content**. This requires creating a challenge table and API endpoints for listing and filtering challenges.
* **4.2 Challenge Submission Flow (File Upload, Caption, Squad Tagging)**
  + **Current Status**: **❌ Missing/Non-functional**. There is no actual mechanism for users to upload files, add captions, or tag squads for challenge submissions.
  + **What Needs to be Addressed**: Implement a full challenge submission flow that includes file upload capabilities and metadata tagging.
* **4.3 PLUR Leaderboard & Challenge Winners**
  + **Current Status**: **⚠️ Partially Implemented (Visual Only)**. A "PLURstreak System" and a visual leaderboard are present. However, this is **not based on real data or gamification logic**.
  + **What Needs to be Addressed**: Integrate a real leaderboard that dynamically updates based on challenge completions and "PLUR" points, and implement logic for determining and showcasing challenge winners.

### **PHASE 5: Marketplace**

The marketplace is currently a **static display** with no functional e-commerce or inventory management.

* **5.1 Payments Integration (Stripe/Other)**
  + **Current Status**: **❌ Missing/Non-functional**. Despite claims in documentation, the "Harsh Reality" assessment and your review explicitly state **"no payment processing despite Stripe integration claims"** and "no Stripe payment integration at all".
  + **What Needs to be Addressed**: Implement Stripe or another payment gateway for real transaction processing.
* **5.2 Product Listings & Inventory Management**
  + **Current Status**: **⚠️ Partially Implemented (Static)**. "Gear Drops of the Day" carousel and product boxes are visually present with fake items. While user-submitted outfit galleries exist on the old site, the new project has **no actual marketplace functionality**.
  + **What Needs to be Addressed**: Enable real product listings and an inventory management system, including support for both physical and digital items, and potentially user-to-user trading.
* **5.3 Order Confirmation, Digital Downloads, Gift Flows**
  + **Current Status**: **❌ Missing/Non-functional**. These post-purchase functionalities are not implemented.
  + **What Needs to be Addressed**: Develop comprehensive order confirmation, digital download delivery, and gift flows.

### **PHASE 6: Community & Crews**

While basic community features (forums) exist on the original EDMShuffle.com, the **advanced social and "crew" features for the new project are largely undeveloped**.

* **6.1 Live Chat System (Real-Time Messaging)**
  + **Current Status**: **⚠️ Partially Implemented (Fake)**. A "Festival Lounge" with a "fake chat feed" of pre-seeded messages is visually present.
  + **What Needs to be Addressed**: Implement a real-time live chat system, including private and group channels.
* **6.2 Squad/Fam Features (Creation, Management, Invites)**
  + **Current Status**: **⚠️ Partially Implemented (Placeholder/Visual)**. A "PLURcrew Builder Sidebar" with visual avatar cards and slots exists. The Archetype Quiz is meant to tie into this, but the crew system is a **placeholder and not interactive**. The "Avatar Summon System" is client-side only.
  + **What Needs to be Addressed**: Develop a gamified squad/crew system with actual creation, management, invitations, member actions, and inter-squad events.
* **6.3 Commenting System**
  + **Current Status**: **⚠️ Partially Implemented (Fake)**. Your review notes "just comments that are kind of feeding through probably presumably people Commenting".
  + **What Needs to be Addressed**: Implement a functional commenting system for user-generated content and platform features.

### **PHASE 7: Admin + Docs**

This phase is critical for the project's long-term manageability and integrity but **remains largely unimplemented**.

* **7.1 Integrated Admin Dashboard**
  + **Current Status**: **❌ Missing/Non-functional**. An admin dashboard is **not implemented**.
  + **What Needs to be Addressed**: Build a comprehensive admin dashboard with role-based access for content, user, event, and product management.
* **7.2 Documentation Pipeline: Root Directory + Microtask Enforcement**
  + **Current Status**: **⚠️ Partially Implemented (Needs Update)**. Various documentation files exist (e.g., project\_overview.md, technical\_architecture.md). However, maintaining consistency and accuracy between claims and reality has been a recurring issue.
  + **What Needs to be Addressed**: Continuously update all project documentation to accurately reflect the current state of features, and enforce consistent adherence to the documentation pipeline for all development tasks.
* **7.3 Integrity & Compliance Automation**
  + **Current Status**: **⚠️ Partially Implemented (Conceptual)**. Integrity rules for LLMs (e.g., no placeholders without labels) have been defined, and tools like Gemini CLI are planned for validation. However, the automated enforcement of these principles is not yet fully in place.
  + **What Needs to be Addressed**: Implement automated testing (Jest/Playwright) and establish a CI/CD pipeline (Netlify/Vercel) to ensure consistent quality and functionality.
* **7.4 Periodic Masterplan & FRS Review**
  + **Current Status**: **❌ Missing/Non-functional**. The Product Requirements Document (PRD) and Functional Requirements Spec (FRS) are high-priority but were noted as "NOT YET WRITTEN".
  + **What Needs to be Addressed**: Establish a process for regular review and update of the Masterplan and FRS to align with actual development progress and evolving requirements.

### **PHASE 8: Built-in Marketing Framework**

This phase is **almost entirely unaddressed** in terms of current implementation.

* **8.1 Integrated Marketing Dashboard**
  + **Current Status**: **❌ Missing/Non-functional**. A marketing dashboard is **not implemented**.
  + **What Needs to be Addressed**: Develop a role-protected marketing dashboard to manage campaign creation, scheduling, and distribution across platforms.
* **8.2 Content Generation & Library**
  + **Current Status**: **❌ Missing/Non-functional**. While AI-generated content and autonomous content curation are part of the vision, the autonomous agents designed to deliver this are **not deployed live** and only produce fake output.
  + **What Needs to be Addressed**: Implement content generation capabilities leveraging LLM endpoints and build a library for generated assets. This is contingent on the core backend and agent functionality being established.
* **8.3 Marketing Best Practices, Failsafes, & Innovation**
  + **Current Status**: **❌ Missing/Non-functional**. While marketing strategies are outlined, the implementation of automation (n8n), approval workflows, and analytics integration within the platform itself is missing.
  + **What Needs to be Addressed**: Build out the automation integration and workflow processes within the marketing dashboard to support effective content deployment and monitoring.

**In summary, while EDM Shuffle has a visually appealing frontend and a clear vision, most of the core functional requirements across all phases, particularly backend integrations, real-time interactivity, and the full implementation of AI/agent-driven features, are still either non-existent, simulated, or only partially implemented cosmetically. The project is currently recognized as being far from its MVP state due to these fundamental gaps**.