🎵 UPRISE NEXT - Conversation Handoff Document

Date Created: November 13, 2025
Repository: https://github.com/ancientagent/UPRISE_NEXT
Project: Music community platform monorepo rebuild
Phase Completed: T1-T4 Foundation

📋 Table of Contents

🎯 Conversation Summary

What Was Accomplished

This conversation successfully established the foundational architecture for the UPRISE NEXT music community platform. We completed the first four critical tasks (T1-T4) that provide:

1. **Architectural Boundaries** - Web-tier contract guard preventing database access from frontend
2. **Geospatial Features** - PostGIS integration for location-based community features
3. **Real-Time Communication** - Socket.IO server with JWT authentication and community namespaces
4. **Testing Infrastructure** - Comprehensive test suites validating all core functionality

All work has been committed to GitHub with proper documentation, and a git force-push recovery was successfully performed to restore lost documentation commits.

Key Achievements

✅ Complete Turborepo monorepo setup
✅ Three applications deployed (web, api, socket)
✅ PostgreSQL with PostGIS configured
✅ JWT authentication implemented
✅ Comprehensive testing infrastructure
✅ Full documentation suite created
✅ Git repository properly managed with recovery performed

🏗️ Current Project State

Repository Structure

```
UPRISE_NEXT/
├── apps/
│   ├── web/                    # Next.js 15 web application
│   │   ├── src/
│   │   │   ├── app/            # App router pages
│   │   │   ├── lib/            # Web-tier guard, API client, utilities
│   │   │   └── middleware.ts   # Request validation middleware
│   │   ├── __tests__/          # Web-tier boundary tests
│   │   └── WEB_TIER_BOUNDARY.md
│   │
│   ├── api/                    # NestJS REST API service
│   │   ├── src/
│   │   │   ├── auth/           # JWT authentication
│   │   │   ├── communities/    # PostGIS endpoints
│   │   │   ├── health/         # Health check endpoints
│   │   │   ├── tracks/         # Track management (stub)
│   │   │   ├── users/          # User management (stub)
│   │   │   └── events/         # Event management (stub)
│   │   ├── prisma/
│   │   │   ├── schema.prisma   # PostGIS-enabled schema
│   │   │   └── migrations/     # Database migrations
│   │   └── test/               # API integration tests
│   │
│   ├── socket/                 # Socket.IO real-time server
│   │   ├── src/
│   │   │   ├── middleware/     # JWT auth middleware
│   │   │   ├── handlers/       # Root namespace handlers
│   │   │   ├── namespaces/     # Community namespace handlers
│   │   │   └── utils/          # Logger utility
│   │   └── test/               # Socket.IO tests
│   │
│   └── workers/
│       └── transcoder/         # FFmpeg media transcoder (stub)
│
├── packages/
│   ├── ui/                     # Shared shadcn/ui components
│   ├── types/                  # Shared TypeScript types
│   └── config/                 # Shared configuration
│
├── docs/                       # Comprehensive documentation
│   ├── AGENT_STRATEGY_AND_HANDOFF.md
│   ├── CHANGELOG.md
│   ├── ENVIRONMENTS.md
│   ├── PHASE1_COMPLETION_REPORT.md
│   ├── PROJECT_STRUCTURE.md
│   ├── RUNBOOK.md
│   ├── STRATEGY_CRITICAL_INFRA_NOTE.md
│   └── Specifications/
│       └── README.md
│
├── README.md                   # Main project documentation
├── T1-T4_IMPLEMENTATION_GUIDE.md  # Detailed implementation guide
├── RECOVERY_SUMMARY.md         # Git recovery documentation
├── docker-compose.yml          # PostgreSQL + PostGIS + Redis
├── turbo.json                  # Turborepo configuration
└── pnpm-workspace.yaml         # Workspace configuration
```

Technology Stack

Layer	Technology	Version	Purpose
Frontend	Next.js	15.x	Server-rendered React application
UI Library	shadcn/ui	Latest	Radix UI + Tailwind components
Styling	Tailwind CSS	3.x	Utility-first CSS framework
Backend API	NestJS	10.x	REST API with modular architecture
Real-time	Socket.IO	4.x	WebSocket-based bi-directional communication
Database	PostgreSQL	15	Primary data store
Geospatial	PostGIS	3.3	GPS and location-based features
ORM	Prisma	5.x	Type-safe database client
Validation	Zod	3.x	Schema validation
Auth	JWT	-	Stateless authentication
Media	FFmpeg	-	Audio/video transcoding
Storage	S3/R2	-	Object storage for media
Testing	Jest	29.x	Unit and integration testing
Monorepo	Turborepo	Latest	Build system and task orchestration
Package Manager	pnpm	9.x	Fast, efficient package manager

Applications Overview

1. apps/web - Next.js Web Application

Status: ✅ Operational with boundary enforcement
Port: 3000 (development)
Deployment: Vercel

Key Features:
- Next.js 15 with App Router
- Server components for SEO
- Web-tier contract guard (prevents direct DB access)
- API client for backend communication
- Socket.IO client integration
- Tailwind CSS + shadcn/ui

Key Files:
- `src/lib/web-tier-guard.ts` - Runtime protection against DB access
- `src/lib/api.ts` - HTTP API client
- `src/lib/socket.ts` - Socket.IO client setup
- `src/middleware.ts` - Request validation
- `WEB_TIER_BOUNDARY.md` - Architecture documentation

2. apps/api - NestJS API Service

Status: ✅ Operational with PostGIS integration
Port: 4000 (development)
Deployment: Fly.io / AWS App Runner

Key Features:
- RESTful API with Zod validation
- PostGIS geospatial queries
- JWT authentication
- Health check endpoints
- CORS configuration
- Structured error handling

Key Endpoints:

```
POST   /api/communities              # Create community with geofence
GET    /api/communities/nearby       # Find nearby communities
POST   /api/communities/:id/verify-location  # Verify user location
GET    /api/health                   # Overall health check
GET    /api/health/postgis           # PostGIS verification
GET    /api/health/db                # Database connection check
```

Key Files:
- `src/communities/communities.service.ts` - PostGIS query logic
- `src/communities/dto/community.dto.ts` - Zod validation schemas
- `src/health/health.service.ts` - Health check implementation
- `prisma/schema.prisma` - Database schema with PostGIS types

3. apps/socket - Socket.IO Server

Status: ✅ Operational with JWT auth and logging
Port: 4001 (development)
Deployment: Fly.io / AWS App Runner

Key Features:
- JWT authentication middleware
- Dynamic community namespaces (`/community/:communityId`)
- Structured logging with context
- Presence tracking
- Graceful shutdown
- Ping/pong health checks

Key Events:

```
// Community Namespace Events
join-community           # Join a community room
leave-community          # Leave a community room
community-message        # Send message to community
track-share              # Share a track
track-reaction           # React to a track
typing:start/stop        # Typing indicators

// Root Namespace Events
presence:update          # Update user status
direct-message           # Send DM to user
notification-read        # Mark notification read
ping/pong                # Health check
```

Key Files:
- `src/middleware/auth.ts` - JWT authentication
- `src/namespaces/communities.ts` - Community event handlers
- `src/handlers/index.ts` - Root namespace handlers
- `src/utils/logger.ts` - Structured logging utility

4. apps/workers/transcoder - Media Transcoding Worker

Status: ⚠️ Stub implementation (not yet functional)
Deployment: AWS Fargate / Fly.io

Planned Features:
- FFmpeg-based audio/video transcoding
- Multiple format output (MP3, AAC, HLS)
- S3/R2 integration
- Progress webhooks
- Queue-based processing

✅ Completed Tasks (T1-T4)

T1: Web-Tier Contract Guard

Objective: Prevent direct database access from the web application layer.

What Was Implemented

1. **Runtime Protection** (`apps/web/src/lib/web-tier-guard.ts`)
   - `isWebTier()` - Detects if code is running in web/browser context
   - `assertNotWebTier()` - Throws error if called from web tier
   - `guardPrismaClient()` - Prevents Prisma Client instantiation
   - `guardDatabaseAccess()` - Blocks any database operations
   - Module loader hooks to catch import attempts

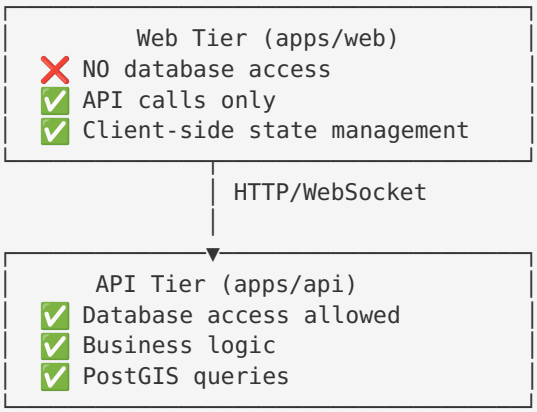
2. **Type-Level Protection** (`apps/web/src/lib/types/web-tier.d.ts`)
   - `ApiOnly<T>` - Type marker for API-tier only values
   - `WebSafe<T>` - Type marker for web-safe values
   - Prisma Client declared as `never` type in web context

3. **ESLint Rules** (`apps/web/.eslintrc.json`)
   - `no-restricted-imports` rule blocks `@prisma/client`
   - Pattern matching for database-related imports
   - Custom error messages

4. **Next.js Middleware** (`apps/web/src/middleware.ts`)
   - Runs on every request
   - Validates headers for suspicious patterns
   - Adds security headers

5. **Comprehensive Documentation** (`apps/web/WEB_TIER_BOUNDARY.md`)
   - Explains architecture rationale
   - Provides correct usage patterns
   - Troubleshooting guide

Architecture Pattern



Testing

- Test file: `apps/web/__tests__/web-tier-guard.test.ts`
- 8 test cases covering all guard functions
- Validates error messages and types

Example Usage

```
// ❌ WRONG - This will be caught and blocked
import { PrismaClient } from '@prisma/client';
const prisma = new PrismaClient();

// ✅ CORRECT - Use API client
import { api } from '@/lib/api';
const communities = await api.get('/communities');
```

T2: PostGIS Integration

Objective: Implement geospatial features for location-based community functionality.

What Was Implemented

1. **Database Schema** (`apps/api/prisma/schema.prisma`)
   - PostGIS extension enabled
   - `geography` columns for lat/lng storage
   - GIST spatial indexes for performance
   - Applied to: Community, Event, User, Track models

2. **Migration** (`apps/api/prisma/migrations/20241113000000_init_postgis/`)
   - `CREATE EXTENSION postgis;`
   - Adds geography columns: `geofence` , `geoboundary` , `location` , `origin`
   - Creates GIST indexes for spatial queries
   - All models have location support

3. **Zod Validation Schemas** (`apps/api/src/communities/dto/community.dto.ts`)
   `typescript`
   `LatitudeSchema # -90 to 90`
   `LongitudeSchema # -180 to 180`
   `RadiusSchema # 10m to 50,000m (50km)`
   `CreateCommunityWithGeoSchema`
   `FindNearbyCommunitiesSchema`
   `VerifyLocationSchema`

4. **PostGIS Endpoints** (`apps/api/src/communities/`)

POST /api/communities
- Creates community with GPS geofence
- Converts lat/lng to PostGIS `POINT` geography
- Stores radius for geofence verification

`bash`
`curl -X POST http://localhost:4000/api/communities \`
`-H "Content-Type: application/json" \`
`-d '{`
`"name": "SF Music Scene",`
`"slug": "sf-music",`
`"lat": 37.7749,`
`"lng": -122.4194,`

```
      "radius": 5000
    }'
```

GET /api/communities/nearby
- Uses `ST_DWithin` for efficient spatial search
- Returns communities sorted by distance
- Calculates distance in meters using `ST_Distance`

```bash
  curl "http://localhost:4000/api/communities/nearby?
lat=37.7749&lng=-122.4194&radius=5000&limit=20"
```

POST /api/communities/:id/verify-location
- Checks if user is within community geofence
- Returns boolean + distance in meters

```bash
  curl -X POST http://localhost:4000/api/communities/123/verify-location \
    -H "Content-Type: application/json" \
    -d '{"lat": 37.7749, "lng": -122.4194}'
```

1. **Health Check** (`apps/api/src/health/`)
   - `GET /api/health/postgis` - Verifies PostGIS extension
   - Tests real spatial calculation (SF to LA distance)
   - Returns PostGIS version and spatial_ref_sys count

```bash
  curl http://localhost:4000/api/health/postgis
```

PostGIS Functions Used

Function	Purpose	Example
`ST_GeogFromText`	Convert lat/lng to geography point	`ST_GeogFromText('POINT(-122.4194 37.7749)')`
`ST_DWithin`	Find points within radius (efficient)	`ST_DWithin(geofence, point, 5000)`
`ST_Distance`	Calculate distance in meters	`ST_Distance(point1, point2)`
`ST_X` , `ST_Y`	Extract coordinates	`ST_X(geofence::geometry)`

Database Configuration

docker-compose.yml:

```
services:
  postgres:
    image: postgis/postgis:15-3.3
    environment:
      POSTGRES_USER: uprise
      POSTGRES_PASSWORD: uprise
      POSTGRES_DB: uprise_dev
    ports:
      - "5432:5432"
```

Start Database:

```
docker-compose up -d
```

Testing

- Test file: `apps/api/test/communities.test.ts`
- 8+ test cases for all PostGIS endpoints
- Validates distance calculations and spatial queries

T3: Real-Time Socket.IO Functionality

Objective: Implement real-time communication for music communities.

What Was Implemented

1. **JWT Authentication Middleware** (`apps/socket/src/middleware/auth.ts`)
   - Validates JWT from `handshake.auth.token` or `query.token`
   - Extracts user data: `{ sub, email, username }`
   - Attaches user to socket: `socket.data.user`
   - Logs all authentication attempts
   - Provides `requireAuth()` helper for protected handlers

```
typescript
  // Client-side authentication
  const socket = io('http://localhost:4001', {
    auth: { token: jwtToken }
  });
```

1. **Structured Logger** (`apps/socket/src/utils/logger.ts`)
   - ISO 8601 timestamps
   - Log levels: INFO, WARN, ERROR, DEBUG
   - Context tracking (socketId, userId, username)
   - Specialized methods:
     - `logConnection()` - Socket connects
     - `logDisconnection()` - Socket disconnects
     - `logEvent()` - Event emissions
     - `logAuthSuccess/Failure()` - Authentication
     - `logJoinRoom/LeaveRoom()` - Room operations

`[2024-11-13T12:00:00.000Z] [INFO] 🔌 Socket connected | socketId: "abc123" userId: "user-456" username: "djsmith"`

`[2024-11-13T12:00:05.000Z] [INFO] 🏠 User joined room | userId: "user-456" room: "community:sf-music"`

1. **Dynamic Community Namespaces** (`apps/socket/src/namespaces/communities.ts`)
   - Pattern: `/community/:communityId`
   - Each community gets isolated namespace
   - JWT required for all connections
   - Automatic presence tracking

Implemented Events:

Event	Direction	Purpose
`join-community`	Client → Server	Explicitly join community room
`leave-community`	Client → Server	Explicitly leave community room
`community-message`	Client → Server	Send message to all members
`track-share`	Client → Server	Share a track with community
`track-reaction`	Client → Server	React to a track
`typing:start`	Client → Server	User started typing
`typing:stop`	Client → Server	User stopped typing
`request-sync`	Client → Server	Request playback state sync
`user:joined`	Server → Clients	User connected to namespace
`user:left`	Server → Clients	User disconnected
`community:member-active`	Server → Clients	User explicitly joined
`community:member-inactive`	Server → Clients	User explicitly left
`community-message:new`	Server → Clients	New message broadcast
`track:shared`	Server → Clients	Track shared notification
`track:reaction`	Server → Clients	Reaction notification

1. **Root Namespace Handlers** (`apps/socket/src/handlers/index.ts`)
   - `presence:update` - User status changes (online/away/offline)
   - `direct-message` - Send DM to another user

   - `notification-read` - Mark notification as read
   - `ping` / `pong` - Keep-alive health check

2. **Server Configuration** (`apps/socket/src/index.ts`)
   - CORS: Configurable origins
   - Transport: WebSocket + polling fallback
   - Ping interval: 25 seconds
   - Ping timeout: 60 seconds
   - Graceful shutdown with 10s timeout
   - Connection stats logged every minute
   - Error handling for uncaught exceptions

Client Usage Example

```
import { io } from 'socket.io-client';

// Connect to community namespace with JWT
const socket = io('http://localhost:4001/community/sf-music', {
  auth: { token: jwtToken }
});

// Listen for connection
socket.on('connect', () => {
  console.log('Connected:', socket.id);
});

// Join community explicitly
socket.emit('join-community', {
  communityId: 'sf-music',
  location: { lat: 37.7749, lng: -122.4194 }
});

// Send message
socket.emit('community-message', {
  content: 'Hello everyone!',
  type: 'text'
});

// Listen for messages
socket.on('community-message:new', (data) => {
  console.log(`${data.username}: ${data.content}`);
});

// Leave community
socket.emit('leave-community', { communityId: 'sf-music' });

// Disconnect
socket.disconnect();
```

Testing

- Test file: `apps/socket/test/socket.test.ts`
- 6+ test cases covering authentication and events
- Validates JWT flow and logging

T4: Comprehensive Test Suites

Objective: Create smoke tests to verify all T1-T3 functionality.

What Was Implemented

1. **Web-Tier Boundary Tests** (`apps/web/__tests__/web-tier-guard.test.ts`)

```
bash
  ✓ isWebTier should detect web tier correctly
  ✓ isWebTier should detect non-web tier correctly
  ✓ assertNotWebTier should throw in web tier
  ✓ guardPrismaClient should throw when accessing PrismaClient
  ✓ guardDatabaseAccess should throw when called in web tier
  ✓ ApiOnly type should make values never in web tier
  ✓ WebSafe type should allow values in web tier
  ✓ api client should be accessible in web tier
```

2. **API Integration Tests** (`apps/api/test/`)

communities.test.ts

```
bash
  ✓ POST /api/communities - create with geofence
  ✓ GET /api/communities/nearby - find nearby
  ✓ POST /api/communities/:id/verify-location - verify user location
  ✓ PostGIS extension should be enabled
  ✓ Distance calculation should be accurate (SF to LA)
  ✓ Validation should reject invalid coordinates
  ✓ Validation should reject invalid radius
  ✓ Should calculate distance in meters correctly
```

health.test.ts

```
bash
  ✓ GET /api/health - should return healthy status
  ✓ GET /api/health/postgis - should verify PostGIS
  ✓ GET /api/health/db - should check database connection
  ✓ PostGIS functionality test should calculate distance
```

1. **Socket.IO Tests** (`apps/socket/test/socket.test.ts`)

```
bash
  ✓ should authenticate with valid JWT
  ✓ should reject invalid JWT
  ✓ should reject connection without JWT
  ✓ should handle join-community event
  ✓ should handle community-message event
  ✓ should handle leave-community event
  ✓ should log connection with user data
  ✓ should log disconnection
```

Test Configuration

Jest Setup:

- `apps/web/jest.config.js` - Environment: `jsdom` (browser simulation)
- `apps/api/jest.config.js` - Environment: `node` (server)
- `apps/socket/jest.config.js` - Environment: `node` (server)

Coverage Targets:
- Web Tier: 80%+ of guards/middleware
- API Tier: 70%+ of controllers/services
- Socket Tier: 70%+ of handlers/middleware

Running Tests

```
# All tests across monorepo
pnpm test

# Specific application
pnpm --filter web test
pnpm --filter api test
pnpm --filter socket test

# Watch mode (auto-rerun on changes)
pnpm --filter web test:watch

# Coverage report
pnpm --filter api test:coverage

# Debug mode
pnpm --filter socket test --verbose
```

Test Output Example

```
 PASS  apps/web/__tests__/web-tier-guard.test.ts
 PASS  apps/api/test/communities.test.ts
 PASS  apps/api/test/health.test.ts
 PASS  apps/socket/test/socket.test.ts

Test Suites: 4 passed, 4 total
Tests:       24 passed, 24 total
Snapshots:   0 total
Time:        8.234s
```

📦 GitHub Status

Repository Information

- **URL:** https://github.com/ancientagent/UPRISE_NEXT
- **Owner:** ancientagent
- **Default Branch:** main
- **Last Updated:** November 13, 2025

Commit History

```
git log --oneline --all

9bf5f6d docs: Add recovery summary for force push incident
d81e81c Organize documentation into /docs hierarchy (RECOVERED)
edbed94 Add implementation guide and tier boundary PDFs
60775ca docs: Add comprehensive T1-T4 implementation guide
c1a51ae docs: Add comprehensive T1-T4 feature documentation
13da88b chore: Update web app layout and globals CSS
472bac7 feat(T4): Add Comprehensive Test Suites
842cdf2 feat(T3): Implement Real-Time Socket.IO with Community Namespaces
3682142 feat(T2): Implement PostGIS API with Geospatial Features
3d32638 feat(T1): Implement Web-Tier Contract Guard
d963518 Initial commit: Complete Turborepo monorepo setup
```

Current Status

```
# Verify repository is clean
cd /home/ubuntu/UPRISE_NEXT
git status
# Output: On branch main
#         Your branch is up to date with 'origin/main'.
#         nothing to commit, working tree clean

# Verify remote
git remote -v
# Output: origin  https://github.com/ancientagent/UPRISE_NEXT.git (fetch)
#         origin  https://github.com/ancientagent/UPRISE_NEXT.git (push)

# Verify all commits are pushed
git log origin/main..HEAD
# Output: (empty - all commits pushed)
```

What's Been Pushed

✅ All T1-T4 implementation code
✅ Comprehensive test suites
✅ Documentation (README, implementation guides)
✅ Configuration files (docker-compose, turbo.json, etc.)
✅ Recovery documentation
✅ Organized /docs hierarchy

🔄 Recovered Work

Force Push Incident

Date: November 13, 2025 at 06:00:20 UTC
Status: ✅ Successfully recovered

What Happened

A force push was made to the `main` branch that overwrote a commit containing organized documentation. The commit was in an unreferenced state on GitHub but not yet garbage collected.

Recovery Process

1. **Detection:** Used GitHub Events API to identify the overwritten commit SHA
2. **Retrieval:** Fetched the lost commit (`d1b80937`) directly from GitHub
3. **Restoration:** Cherry-picked the commit into the current branch
4. **Verification:** Confirmed all files were restored correctly
5. **Documentation:** Created `RECOVERY_SUMMARY.md` to document the incident

Recovered Commit

SHA: `d1b80937fedd03059bb688e5fd04df80831ddb89`
Author: baris bariseman@gmail.com
Date: Wed Nov 12 23:54:47 2025 -0600
Message: “Organize documentation into /docs hierarchy”

Recovered Files

The following 8 files were restored to the `/docs` directory:

1. `docs/AGENT_STRATEGY_AND_HANDOFF.md` - Agent workflow strategies
2. `docs/CHANGELOG.md` - Project change log
3. `docs/ENVIRONMENTS.md` - Environment setup guide
4. `docs/PHASE1_COMPLETION_REPORT.md` - Phase 1 completion status
5. `docs/PROJECT_STRUCTURE.md` - Repository structure documentation
6. `docs/RUNBOOK.md` - Operational procedures
7. `docs/STRATEGY_CRITICAL_INFRA_NOTE.md` - Infrastructure notes
8. `docs/Specifications/README.md` - Specification index

Total Changes: 684 insertions, 8 files

Prevention Recommendations

1. Use `git push --force-with-lease` instead of `git push --force`
2. Always `git fetch` before force pushing
3. Review remote commits before force operations
4. Consider branch protection rules on GitHub

Full Recovery Documentation

See `RECOVERY_SUMMARY.md` and `RECOVERY_SUMMARY.pdf` for complete details.

🔮 Pending Work (T5-T8)

The following tasks were mentioned but **NOT implemented yet**. They require clarification and should be prioritized in the next conversation.

T5: Advanced Real-Time Features

Status: ⚠️ Not Started - Needs Requirements

Potential Features:
- Synchronized music playback across users
- Live DJ session functionality

- Collaborative playlists with real-time updates
- Voice chat integration
- Screen sharing for music production
- Real-time collaborative mixing board
- Live voting/polling for track selection

Questions to Answer:
1. What specific real-time features are highest priority?
2. Should synchronized playback use a coordinator pattern?
3. Do we need WebRTC for voice/video, or just data channels?
4. How should latency be handled for live DJ sessions?
5. What's the expected concurrent user limit per community?

Dependencies:
- ✅ Socket.IO infrastructure (T3 complete)
- ⚠️ Media player integration (not yet specified)
- ⚠️ User presence system (basic version in T3)

T6: Media Transcoding Pipeline

Status: ⚠️ Stub Exists - Needs Implementation

Current State:
- `apps/workers/transcoder` directory exists
- No functional code implemented
- FFmpeg dependency documented

Planned Features:
- Asynchronous audio transcoding
- Multiple output formats (MP3, AAC, FLAC, HLS)
- Multiple quality levels (128k, 256k, 320k)
- Waveform generation
- Metadata extraction
- S3/Cloudflare R2 storage integration
- Progress tracking via webhooks/events
- Queue-based processing (Bull/BullMQ)

Questions to Answer:
1. What audio formats should be supported for input?
2. What output formats are required? (MP3, AAC, FLAC, HLS, etc.)
3. Should we generate waveforms for visualization?
4. What storage service? (AWS S3, Cloudflare R2, DigitalOcean Spaces)
5. How should progress be reported? (Webhooks, Socket.IO, polling)
6. What's the expected transcoding volume (tracks per day)?
7. Should we use a queue system? (Bull, BullMQ, SQS)

Technical Decisions Needed:
- Worker deployment strategy (AWS Fargate, Fly.io, k8s)
- Scaling approach (horizontal vs. vertical)
- File size limits

- Timeout handling
- Error recovery strategy

T7: Event System

Status: ⚠️ Not Started - Needs Specification

Current State:
- Basic Event model exists in Prisma schema
- No endpoints implemented
- No business logic

Potential Features:
- Create events with geofenced boundaries
- RSVP system with capacity limits
- Event check-in using GPS verification
- Lineup management (DJs, performers)
- Schedule/timeslot management
- Event discovery (nearby, by genre, by date)
- Ticket integration (free or paid)
- Event chat/community namespace
- Live event updates and notifications
- Post-event recaps and media sharing

Questions to Answer:
1. Should events use the same PostGIS geofencing as communities?
2. Do we need ticket/payment integration? (Stripe, PayPal)
3. Should events have their own Socket.IO namespaces?
4. What's the relationship between events and communities?
5. Do we need event analytics? (attendance, engagement)
6. Should users receive reminders/notifications?

API Endpoints to Design:

```
POST   /api/events                    # Create event
GET    /api/events/nearby             # Find nearby events
GET    /api/events/:id                # Get event details
POST   /api/events/:id/rsvp           # RSVP to event
POST   /api/events/:id/check-in       # GPS-verified check-in
GET    /api/events/:id/attendees      # List attendees
PUT    /api/events/:id/lineup         # Update lineup
DELETE /api/events/:id                # Cancel event
```

T8: Analytics and Monitoring

Status: ⚠️ Not Started - Needs Configuration

Documented Tools:
- Sentry (error tracking)

- PostHog (product analytics)
- Custom metrics (Prometheus/Grafana potential)

What Needs to be Implemented:

1. **Error Tracking (Sentry)**
   - Install `@sentry/nextjs` for web app
   - Install `@sentry/node` for API and Socket servers
   - Configure error capture and reporting
   - Set up release tracking
   - Configure source maps for production
   - Alert configuration

2. **Product Analytics (PostHog)**
   - Install PostHog client libraries
   - Event tracking strategy:
     - User signups/logins
     - Community joins/leaves
     - Track plays/shares
     - Event RSVPs/check-ins
     - Message sends
     - User property tracking
     - Feature flag setup (A/B testing)
     - Funnel analysis configuration

3. **Application Metrics**
   - API response times
   - Database query performance
   - Socket.IO connection counts
   - Transcoder queue length
   - PostGIS query latency
   - Error rates by endpoint

4. **Health Dashboards**
   - Real-time service status
   - Database connection pools
   - Memory/CPU usage
   - API rate limiting metrics
   - WebSocket connection counts

Questions to Answer:
1. Do we have Sentry and PostHog accounts set up?
2. What events are most important to track?
3. What alerts should be configured?
4. Should we use custom metrics infrastructure? (Prometheus, Datadog, etc.)
5. What retention policies for logs/metrics?

Configuration Needed:

```
# Environment variables to add
SENTRY_DSN="..."
SENTRY_ORG="..."
SENTRY_PROJECT="..."
POSTHOG_API_KEY="..."
POSTHOG_HOST="..."
```

💡 Important Context

Architectural Decisions

1. Web-Tier Boundary Enforcement (T1)

Decision: Strict separation between web and data tiers
Rationale:
- Prevents accidental database calls from client-side code
- Reduces bundle size (no Prisma in web app)
- Improves security (database credentials never exposed)
- Forces proper API-driven architecture
- Facilitates future microservices migration

Enforcement Layers:
1. Runtime guards (throws errors)
2. TypeScript types (compile-time checks)
3. ESLint rules (linting errors)
4. Middleware validation (request-level checks)

Trade-offs:
- ✅ Pro: Strong architectural boundaries
- ✅ Pro: Better security and performance
- ⚠️ Con: Slightly more verbose (API calls instead of direct queries)
- ⚠️ Con: Requires API endpoints for all data operations

2. PostGIS for Geospatial Features (T2)

Decision: Use PostGIS extension instead of application-level GPS calculations
Rationale:
- Database-level spatial indexing (GIST) for performance
- Accurate geodesic distance calculations (accounts for Earth's curvature)
- Industry-standard geospatial queries
- Efficient `ST_DWithin` for "nearby" searches
- Foundation for advanced features (polygon boundaries, routing)

Performance Benefits:
- GIST index makes spatial queries ~100x faster than brute-force
- Single query instead of application-level filtering
- Distance calculations in meters (no manual conversions)

Trade-offs:
- ✅ Pro: Excellent performance for spatial queries
- ✅ Pro: Accurate distance calculations

- ⚠️ Con: Requires PostGIS extension (adds complexity)
- ⚠️ Con: Database migration dependency

3. Dynamic Socket.IO Namespaces (T3)

Decision: Use namespaces per community (`/community/:id`) instead of rooms in root namespace
Rationale:
- Better isolation between communities
- Easier to implement per-community middleware
- Simpler authorization logic
- Scales better with many communities
- Facilitates community-specific features

Pattern:

```
/                              # Root namespace (DMs, presence)
/community/sf-music            # SF Music Scene community
/community/la-underground      # LA Underground community
/event/summer-rave-2025        # Event-specific namespace (future)
```

Trade-offs:
- ✅ Pro: Better isolation and authorization
- ✅ Pro: Easier to scale (can shard by namespace)
- ⚠️ Con: Slightly more client-side code (multiple connections)
- ⚠️ Con: More memory per namespace (negligible)

4. Turborepo Monorepo Structure

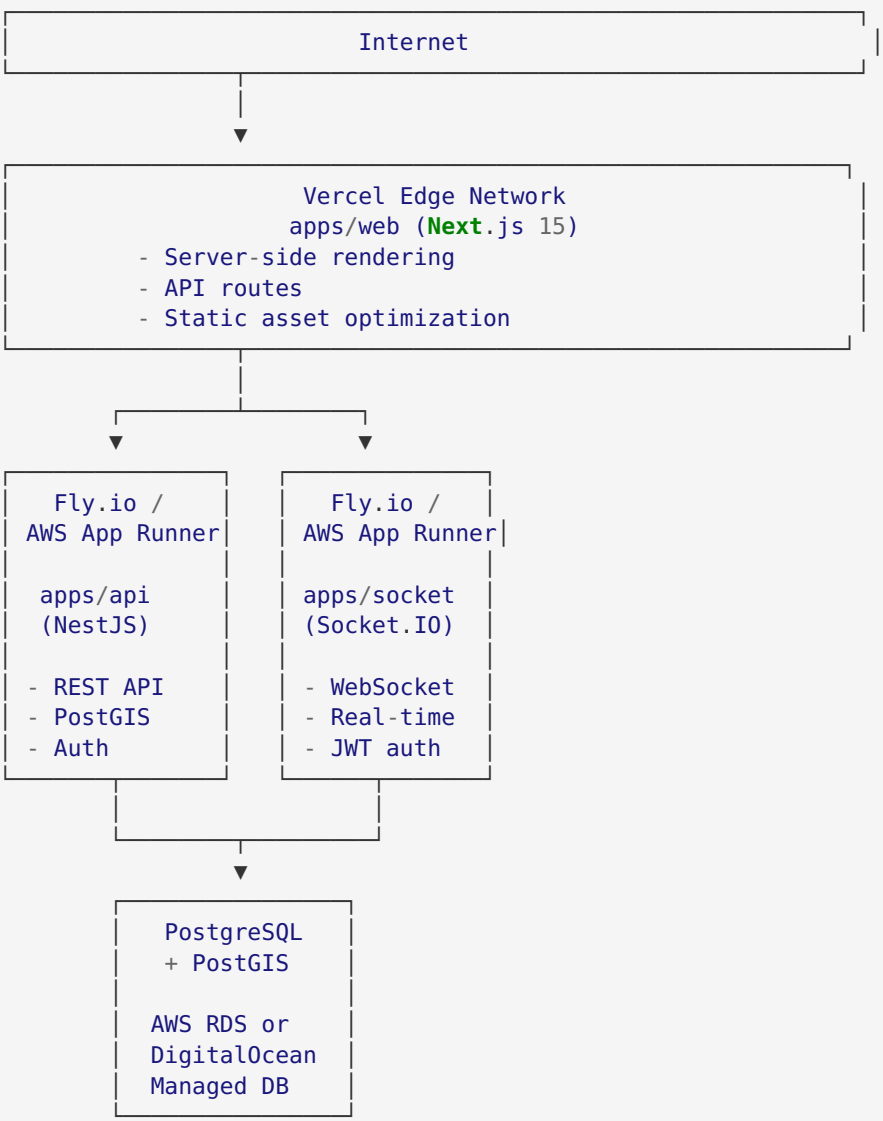
Decision: Use Turborepo for monorepo management
Rationale:
- Shared code between applications (types, UI components)
- Unified dependency management (single pnpm install)
- Efficient builds with caching
- Task orchestration (dev, build, test)
- Better than Nx for TypeScript-focused projects

Benefits:
- Fast incremental builds
- Remote caching (future: Turborepo cache)
- Workspace-aware tasks
- Consistent development experience

Trade-offs:
- ✅ Pro: Faster builds and better DX
- ✅ Pro: Shared code without npm packages
- ⚠️ Con: More complex initial setup
- ⚠️ Con: Requires learning Turborepo patterns

Deployment Targets

Production Architecture



Deployment Commands

Web (Vercel):

```
# Automatic deployment via GitHub integration
git push origin main

# Manual deployment
cd apps/web
vercel --prod
```

API (Fly.io):

```
# Create app
fly apps create uprise-api

# Deploy
cd apps/api
fly deploy

# Scale
fly scale count 2
fly scale vm shared-cpu-2x
```

Socket (Fly.io):

```
# Create app
fly apps create uprise-socket

# Deploy
cd apps/socket
fly deploy

# Scale
fly scale count 2
```

Database:
- Use managed PostgreSQL service (AWS RDS, DigitalOcean, Render)
- Ensure PostGIS extension is enabled
- Configure backups and point-in-time recovery
- Set up read replicas for scaling (future)

Environment Configuration

Required Environment Variables

apps/web (.env.local):

```
# API connection
NEXT_PUBLIC_API_URL="http://localhost:4000"
NEXT_PUBLIC_SOCKET_URL="http://localhost:4001"

# Public keys (safe to expose)
NEXT_PUBLIC_SENTRY_DSN="https://...@sentry.io/..."
NEXT_PUBLIC_POSTHOG_KEY="phc_..."
NEXT_PUBLIC_POSTHOG_HOST="https://app.posthog.com"

# Server-side only (not in NEXT_PUBLIC_)
SENTRY_AUTH_TOKEN="..."
DATABASE_URL="postgresql://..."  # For API routes (future)
```

apps/api (.env):

```
# Database (must have PostGIS)
DATABASE_URL="postgresql://user:pass@localhost:5432/uprise_dev"

# Authentication
JWT_SECRET="your-super-secret-jwt-key-change-this-in-production"
JWT_EXPIRES_IN="7d"

# Server configuration
PORT=4000
NODE_ENV="development"
CORS_ORIGIN="http://localhost:3000"

# External services (T6-T8)
AWS_REGION="us-west-2"
AWS_ACCESS_KEY_ID="..."
AWS_SECRET_ACCESS_KEY="..."
S3_BUCKET_NAME="uprise-media"

# Monitoring (T8)
SENTRY_DSN="https://...@sentry.io/..."
SENTRY_ENVIRONMENT="development"
```

apps/socket (.env):

```
# Must match API JWT secret
JWT_SECRET="your-super-secret-jwt-key-change-this-in-production"

# Server configuration
PORT=4001
NODE_ENV="development"
CORS_ORIGIN="http://localhost:3000"

# Logging
LOG_LEVEL="info"

# Monitoring (T8)
SENTRY_DSN="https://...@sentry.io/..."
```

docker-compose.yml (Local Development):

```
# Included in repository
# Provides PostgreSQL + PostGIS + Redis
docker-compose up -d
```

Security Considerations

1. JWT Authentication

- **Current State:** Implemented for Socket.IO, partially for API
- **Token Format:**

```
json
  {
    "sub": "user_id",
    "email": "user@example.com",
```

```
    "username": "djsmith",
    "iat": 1699900000,
    "exp": 1700500000
  }
```

- **Expiration:** 7 days default
- **Refresh Tokens:** Not yet implemented (T5-T8)
- **Storage:** Client-side (localStorage or memory)

2. CORS Configuration

- **Web Tier:** Configured in Vercel
- **API Tier:** NestJS CORS middleware
- **Socket Tier:** Socket.IO CORS options
- **Production:** Whitelist specific domains only

3. Rate Limiting

- **Status:** Not yet implemented
- **Recommendation:** Use `@nestjs/throttler` for API
- **Future:** Implement per-user rate limits based on JWT

4. Input Validation

- **Web Tier:** Client-side validation (user experience)
- **API Tier:** Zod schemas (authoritative validation)
- **Socket Tier:** Validate all incoming events

5. Database Security

- **Prisma Client:** Parameterized queries (SQL injection protection)
- **PostGIS:** Use Prisma's type-safe queries, not raw SQL
- **Credentials:** Never commit to git, use environment variables

---

🚀 Next Steps

Immediate Actions for Next Conversation

1. **Clarify T5-T8 Requirements**
   - Review the "Pending Work" section above
   - Answer the questions listed for each task
   - Prioritize which features to implement first

2. **Choose Next Task to Implement**
   - **Option A:** T5 (Advanced Real-Time) - Build on existing Socket.IO
   - **Option B:** T6 (Media Transcoding) - Critical for music platform
   - **Option C:** T7 (Event System) - Complete the community features
   - **Option D:** T8 (Analytics/Monitoring) - Set up observability

3. **Review This Handoff Document**
   - Ensure all information is accurate
   - Add any missing context
   - Confirm deployment strategy

Recommended Implementation Order

Option 1: Feature-First Approach

```
T6 (Media Transcoding) → T7 (Event System) → T5 (Advanced Real-Time) → T8 (Monitoring)
```

Rationale: Core music features first, then community features, then observability

Option 2: Infrastructure-First Approach

```
T8 (Monitoring) → T6 (Media Transcoding) → T7 (Event System) → T5 (Advanced Real-Time)
```

Rationale: Set up observability before adding complex features

Option 3: User-Value-First Approach

```
T7 (Event System) → T5 (Advanced Real-Time) → T6 (Media Transcoding) → T8 (Monitoring)
```

Rationale: Deliver visible user features quickly, infrastructure later

Questions to Answer Before Proceeding

1. **What's the highest priority feature for the platform?**
   - Is it events, media transcoding, or real-time features?

2. **Are there any external dependencies or integrations needed?**
   - Payment processing (Stripe)?
   - Cloud storage (AWS S3, Cloudflare R2)?
   - Email service (SendGrid, Postmark)?
   - SMS service (Twilio)?

3. **What's the timeline for launch?**
   - MVP launch date?
   - Beta testing period?
   - Public launch?

4. **What's the expected scale?**
   - Number of users (Day 1, Month 1, Year 1)?
   - Number of communities?
   - Number of events?
   - Media storage requirements?

5. **Are there any design/UI requirements?**
   - Figma designs?
   - Brand guidelines?
   - Accessibility requirements?

🗂️ Quick Reference

Development Commands

```
# Start all services
pnpm dev

# Start specific app
pnpm --filter web dev         # http://localhost:3000
pnpm --filter api dev         # http://localhost:4000
pnpm --filter socket dev      # http://localhost:4001

# Database
docker-compose up -d          # Start PostgreSQL + PostGIS
docker-compose down           # Stop database
docker-compose logs -f        # View logs

# Prisma commands
cd apps/api
pnpm prisma generate          # Generate Prisma Client
pnpm prisma migrate dev       # Run migrations
pnpm prisma studio            # Open database GUI

# Testing
pnpm test                     # Run all tests
pnpm --filter web test        # Test web app
pnpm --filter api test        # Test API
pnpm --filter socket test     # Test socket server
pnpm test:watch               # Watch mode
pnpm test:coverage            # Coverage report

# Build
pnpm build                    # Build all apps
pnpm --filter web build       # Build web app only

# Type checking
pnpm typecheck                # Type check all apps

# Linting
pnpm lint                     # Lint all code
pnpm lint:fix                 # Auto-fix lint errors

# Clean
pnpm clean                    # Remove node_modules and build artifacts
```

Key File Locations

Configuration Files

```
├── turbo.json                          # Turborepo task pipeline
├── pnpm-workspace.yaml                 # Workspace configuration
├── docker-compose.yml                  # Local database setup
├── tsconfig.base.json                  # Base TypeScript config
└── apps/
    ├── web/
    │   ├── .eslintrc.json             # Web linting rules
    │   ├── jest.config.js             # Web test config
    │   ├── next.config.js             # Next.js configuration
    │   └── tailwind.config.ts         # Tailwind CSS config
    ├── api/
    │   ├── jest.config.js             # API test config
    │   ├── nest-cli.json              # NestJS CLI config
    │   └── tsconfig.json              # API TypeScript config
    └── socket/
        ├── jest.config.js             # Socket test config
        └── tsconfig.json              # Socket TypeScript config
```

Core Implementation Files

```
apps/
├── web/src/
│   ├── lib/
│   │   ├── web-tier-guard.ts        # T1: Runtime guard
│   │   ├── api.ts                   # HTTP API client
│   │   └── socket.ts                # Socket.IO client
│   ├── middleware.ts                 # T1: Request validation
│   └── app/                          # Next.js pages
│
├── api/src/
│   ├── communities/
│   │   ├── communities.service.ts   # T2: PostGIS queries
│   │   ├── communities.controller.ts
│   │   └── dto/community.dto.ts     # T2: Zod schemas
│   ├── health/
│   │   ├── health.service.ts        # T2: Health checks
│   │   └── health.controller.ts
│   └── prisma/
│       └── prisma.service.ts        # Prisma connection
│
└── socket/src/
    ├── middleware/auth.ts            # T3: JWT auth
    ├── namespaces/communities.ts     # T3: Community events
    ├── handlers/index.ts             # T3: Root events
    └── utils/logger.ts               # T3: Logging
```

Test Files

```
apps/
├── web/__tests__/
│   └── web-tier-guard.test.ts       # T1 tests
├── api/test/
│   ├── communities.test.ts          # T2 tests
│   └── health.test.ts               # T2 health tests
└── socket/test/
    └── socket.test.ts               # T3 tests
```

Documentation

```
├── README.md                          # Main project documentation
├── T1-T4_IMPLEMENTATION_GUIDE.md      # Detailed implementation guide
├── CONVERSATION_HANDOFF.md            # This document
├── RECOVERY_SUMMARY.md                # Git recovery documentation
├── apps/web/WEB_TIER_BOUNDARY.md      # T1 architecture guide
└── docs/
    ├── AGENT_STRATEGY_AND_HANDOFF.md
    ├── CHANGELOG.md
    ├── ENVIRONMENTS.md
    ├── PHASE1_COMPLETION_REPORT.md
    ├── PROJECT_STRUCTURE.md
    ├── RUNBOOK.md
    └── STRATEGY_CRITICAL_INFRA_NOTE.md
```

Important URLs

Development

- **Web App:** http://localhost:3000
- **API Server:** http://localhost:4000
- **Socket.IO:** http://localhost:4001
- **Prisma Studio:** http://localhost:5555 (run `pnpm prisma studio`)

API Endpoints

```
# Health checks
GET  http://localhost:4000/api/health
GET  http://localhost:4000/api/health/postgis
GET  http://localhost:4000/api/health/db

# Communities (T2)
POST http://localhost:4000/api/communities
GET  http://localhost:4000/api/communities/nearby?lat=37.7749&lng=-122.4194&radius=500
0
POST http://localhost:4000/api/communities/:id/verify-location
```

Socket.IO Namespaces

```
// Root namespace
io('http://localhost:4001')

// Community namespace
io('http://localhost:4001/community/sf-music')
io('http://localhost:4001/community/la-underground')
```

Production (Future)

- **Web:** https://uprise.vercel.app (or custom domain)
- **API:** https://api.uprise.com
- **Socket:** https://socket.uprise.com

Git Workflow

```
# Check status
git status

# View recent commits
git log --oneline -10

# Create feature branch
git checkout -b feature/t5-advanced-realtime

# Stage and commit
git add .
git commit -m "feat(T5): Implement synchronized playback"

# Push to GitHub
git push origin feature/t5-advanced-realtime

# Switch back to main
git checkout main

# Pull latest changes
git pull origin main

# Merge feature branch
git merge feature/t5-advanced-realtime

# Push to main
git push origin main

# View all branches
git branch -a

# Delete feature branch
git branch -d feature/t5-advanced-realtime
```

Database Access

```
# Using Docker
docker exec -it uprise_postgres psql -U uprise -d uprise_dev

# Direct connection
psql postgresql://uprise:uprise@localhost:5432/uprise_dev

# Common SQL commands
\dt                    # List tables
\d+ communities        # Describe communities table
SELECT PostGIS_Version();  # Check PostGIS version
SELECT COUNT(*) FROM spatial_ref_sys;  # Verify PostGIS

# Test PostGIS
SELECT ST_Distance(
  ST_GeogFromText('POINT(-122.4194 37.7749)'),  -- SF
  ST_GeogFromText('POINT(-118.2437 34.0522)')   -- LA
);
-- Expected: ~559465 meters (559 km)
```

Troubleshooting

Problem: PostGIS extension not found

```
# Solution: Enable extension in database
docker exec -it uprise_postgres psql -U uprise -d uprise_dev
CREATE EXTENSION IF NOT EXISTS postgis;
SELECT PostGIS_Version();
```

Problem: Web-tier boundary violation error

```
// Error: WEB-TIER BOUNDARY VIOLATION
// Solution: Remove database imports, use API client
import { api } from '@/lib/api';  // ✅ Correct
// import { prisma } from '@/lib/db';  // ❌ Wrong
```

Problem: Socket.IO authentication fails

```
# Check JWT_SECRET matches between api and socket
# apps/api/.env
JWT_SECRET="same-secret-here"

# apps/socket/.env
JWT_SECRET="same-secret-here"
```

Problem: Tests fail with “Can’t reach database”

```
# Start database first
docker-compose up -d

# Run migrations
cd apps/api && pnpm prisma migrate dev

# Then run tests
pnpm test
```

Problem: Port already in use

```
# Find process using port
lsof -ti:3000  # Replace with your port

# Kill process
kill -9 $(lsof -ti:3000)

# Or change port in package.json
"dev": "next dev -p 3001"
```

📞 Additional Resources

Documentation Links

- **Next.js 15:** https://nextjs.org/docs
- **NestJS:** https://docs.nestjs.com
- **Socket.IO:** https://socket.io/docs/v4
- **Prisma:** https://www.prisma.io/docs
- **PostGIS:** https://postgis.net/documentation
- **Tailwind CSS:** https://tailwindcss.com/docs
- **shadcn/ui:** https://ui.shadcn.com
- **Zod:** https://zod.dev
- **Turborepo:** https://turbo.build/repo/docs
- **pnpm:** https://pnpm.io

Repository Links

- **GitHub:** https://github.com/ancientagent/UPRISE_NEXT
- **Issues:** https://github.com/ancientagent/UPRISE_NEXT/issues
- **Pull Requests:** https://github.com/ancientagent/UPRISE_NEXT/pulls

---

✅ Handoff Checklist

Before starting the next conversation, ensure:

- [ ] This handoff document has been reviewed
- [ ] Git repository is up to date (`git pull origin main`)
- [ ] All services can start successfully (`pnpm dev`)
- [ ] All tests pass (`pnpm test`)
- [ ] Database is running (`docker-compose up -d`)
- [ ] PostGIS is verified (`curl localhost:4000/api/health/postgis`)
- [ ] T5-T8 priorities have been decided
- [ ] Any new requirements are documented
- [ ] Environment variables are configured

Document Version: 1.0
Last Updated: November 13, 2025
Next Review: Before starting T5-T8 implementation

This handoff document provides complete context for continuing the UPRISE NEXT project. It should be referenced at the start of each new conversation to maintain continuity and ensure no work is duplicated or lost.