Based on the verification report, here's a corrected and finalized prompt that addresses all critical issues:

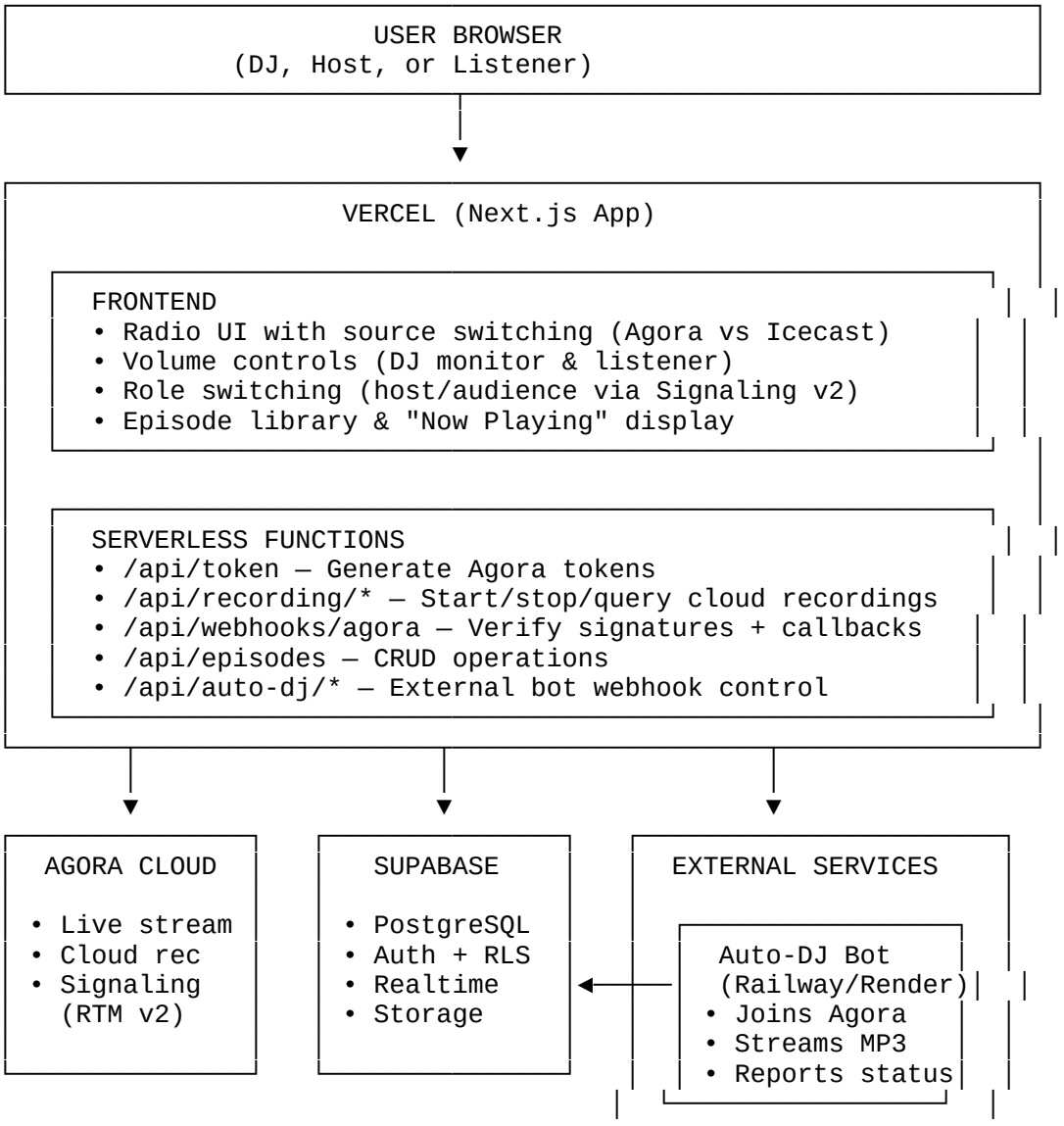
Live Community Radio Platform - Corrected Implementation Prompt

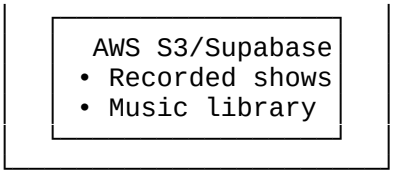
Project Overview

A full-stack community radio platform with dual-mode audio delivery (Live DJ + Auto-DJ), built on Next.js (Vercel), Supabase, and Agora RTC. The platform supports live broadcasting, automated playlist playback, cloud recording, and listener engagement features.

Architecture Overview

System Components





Key Architectural Decisions

1. **Dual Audio Sources**: Live (Agora RTC) and Auto-DJ (Icecast MP3 stream)
2. **Persistent Auto-DJ Bot**: Deployed externally (not on Vercel) for continuous playback
3. **Cloud Recording**: Complete 3-step Agora API flow
4. **Secure Webhooks**: Signature verification for all Agora callbacks

---

Database Schema (Supabase PostgreSQL)

```
-- ============================================
-- ENABLE EXTENSIONS
-- ============================================
CREATE EXTENSION IF NOT EXISTS "uuid-ossp";

-- ============================================
-- USERS TABLE (extends Supabase Auth)
-- ============================================
CREATE TABLE users (
  id UUID PRIMARY KEY REFERENCES auth.users(id) ON DELETE CASCADE,
  email TEXT UNIQUE NOT NULL,
  username TEXT UNIQUE NOT NULL,
  display_name TEXT,
  role TEXT DEFAULT 'listener' CHECK (role IN ('admin', 'dj', 'host',
'listener')),
  avatar_url TEXT,
  created_at TIMESTAMPTZ DEFAULT NOW(),
  updated_at TIMESTAMPTZ DEFAULT NOW()
);

CREATE INDEX idx_users_role ON users(role);
CREATE INDEX idx_users_username ON users(username);

-- ============================================
-- CHANNELS TABLE (State Management)
-- ============================================
CREATE TABLE channels (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  name TEXT UNIQUE NOT NULL,
  slug TEXT UNIQUE NOT NULL,
  description TEXT,
  current_mode TEXT DEFAULT 'auto-dj' CHECK (current_mode IN ('live', 'auto-dj',
'off')),
  current_show_id UUID,
  current_track_id UUID,
  listener_count INTEGER DEFAULT 0,
  is_recording BOOLEAN DEFAULT false,
  recording_resource_id TEXT,
  recording_sid TEXT,
  updated_at TIMESTAMPTZ DEFAULT NOW()
);
```

```
CREATE INDEX idx_channels_slug ON channels(slug);

-- ============================================
-- SHOWS TABLE
-- ============================================
CREATE TABLE shows (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  title TEXT NOT NULL,
  host_id UUID REFERENCES users(id) ON DELETE SET NULL,
  description TEXT,
  channel_id UUID REFERENCES channels(id) ON DELETE CASCADE NOT NULL,
  scheduled_start TIMESTAMPTZ,
  scheduled_end TIMESTAMPTZ,
  is_live BOOLEAN DEFAULT false,
  mode TEXT DEFAULT 'live' CHECK (mode IN ('live', 'auto-dj', 'hybrid')),
  thumbnail_url TEXT,
  created_at TIMESTAMPTZ DEFAULT NOW(),
  updated_at TIMESTAMPTZ DEFAULT NOW()
);

CREATE INDEX idx_shows_host ON shows(host_id);
CREATE INDEX idx_shows_channel ON shows(channel_id);
CREATE INDEX idx_shows_scheduled ON shows(scheduled_start, scheduled_end);

-- ============================================
-- SCHEDULE TABLE
-- ============================================
CREATE TABLE schedule (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  show_id UUID REFERENCES shows(id) ON DELETE CASCADE,
  day_of_week INTEGER CHECK (day_of_week BETWEEN 0 AND 6),
  start_time TIME NOT NULL,
  end_time TIME NOT NULL,
  timezone TEXT DEFAULT 'UTC',
  mode TEXT NOT NULL CHECK (mode IN ('live', 'auto-dj', 'hybrid')),
  is_active BOOLEAN DEFAULT true,
  created_at TIMESTAMPTZ DEFAULT NOW(),
  updated_at TIMESTAMPTZ DEFAULT NOW()
);

CREATE INDEX idx_schedule_day ON schedule(day_of_week, start_time);

-- ============================================
-- TRACKS TABLE (Music Library)
-- ============================================
CREATE TABLE tracks (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  title TEXT NOT NULL,
  artist TEXT NOT NULL,
  album TEXT,
  duration INTEGER NOT NULL, -- in seconds
  storage_path TEXT NOT NULL, -- S3/Supabase Storage path
  storage_url TEXT, -- cached public URL
  license_type TEXT DEFAULT 'royalty-free' CHECK (license_type IN ('royalty-
free', 'licensed', 'community', 'original')),
  license_reference TEXT,
  genre TEXT,
  play_count INTEGER DEFAULT 0,
  last_played_at TIMESTAMPTZ,
  uploaded_by UUID REFERENCES users(id) ON DELETE SET NULL,
  is_active BOOLEAN DEFAULT true,
  created_at TIMESTAMPTZ DEFAULT NOW()
);
```

```
CREATE INDEX idx_tracks_active ON tracks(is_active);
CREATE INDEX idx_tracks_uploaded_by ON tracks(uploaded_by);

-- ============================================
-- PLAYLISTS TABLE
-- ============================================
CREATE TABLE playlists (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  name TEXT NOT NULL,
  description TEXT,
  is_default BOOLEAN DEFAULT false,
  is_fallback BOOLEAN DEFAULT false,
  created_by UUID REFERENCES users(id) ON DELETE SET NULL,
  created_at TIMESTAMPTZ DEFAULT NOW()
);

-- ============================================
-- PLAYLIST_TRACKS TABLE
-- ============================================
CREATE TABLE playlist_tracks (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  playlist_id UUID REFERENCES playlists(id) ON DELETE CASCADE,
  track_id UUID REFERENCES tracks(id) ON DELETE CASCADE,
  sort_order INTEGER NOT NULL DEFAULT 0,
  added_by UUID REFERENCES users(id) ON DELETE SET NULL,
  added_at TIMESTAMPTZ DEFAULT NOW(),
  UNIQUE(playlist_id, track_id, sort_order)
);

CREATE INDEX idx_playlist_tracks_order ON playlist_tracks(playlist_id,
sort_order);

-- ============================================
-- EPISODES TABLE
-- ============================================
CREATE TABLE episodes (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  show_id UUID REFERENCES shows(id) ON DELETE SET NULL,
  title TEXT NOT NULL,
  description TEXT,
  channel_id UUID REFERENCES channels(id) ON DELETE SET NULL,
  audio_url TEXT, -- HLS/m3u8 or MP3 URL
  audio_storage_path TEXT,
  duration INTEGER, -- in seconds
  file_size BIGINT,
  mode TEXT NOT NULL CHECK (mode IN ('live', 'auto-dj')),
  recording_resource_id TEXT,
  recording_sid TEXT,
  listeners_peak INTEGER DEFAULT 0,
  listeners_avg INTEGER DEFAULT 0,
  is_published BOOLEAN DEFAULT true,
  published_at TIMESTAMPTZ,
  created_at TIMESTAMPTZ DEFAULT NOW()
);

CREATE INDEX idx_episodes_show ON episodes(show_id);
CREATE INDEX idx_episodes_channel ON episodes(channel_id);
CREATE INDEX idx_episodes_published ON episodes(is_published, published_at);

-- ============================================
-- RECORDINGS TABLE
-- ============================================
CREATE TABLE recordings (
```

```
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  episode_id UUID REFERENCES episodes(id) ON DELETE CASCADE,
  channel_name TEXT NOT NULL,
  resource_id TEXT NOT NULL,
  sid TEXT NOT NULL,
  status TEXT DEFAULT 'starting' CHECK (status IN ('starting', 'recording',
'stopped', 'failed')),
  file_list JSONB, -- Agora file list response
  storage_vendor INTEGER, -- 1=AWS, 2=Aliyun, etc.
  started_at TIMESTAMPTZ DEFAULT NOW(),
  stopped_at TIMESTAMPTZ,
  created_at TIMESTAMPTZ DEFAULT NOW()
);

CREATE INDEX idx_recordings_sid ON recordings(sid);
CREATE INDEX idx_recordings_episode ON recordings(episode_id);

-- ============================================
-- SPEAKING_REQUESTS TABLE
-- ============================================
CREATE TABLE speaking_requests (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  show_id UUID REFERENCES shows(id) ON DELETE CASCADE,
  user_id UUID REFERENCES users(id) ON DELETE CASCADE,
  username TEXT NOT NULL,
  message TEXT,
  status TEXT DEFAULT 'pending' CHECK (status IN ('pending', 'approved',
'denied', 'cancelled')),
  requested_at TIMESTAMPTZ DEFAULT NOW(),
  resolved_at TIMESTAMPTZ
);

CREATE INDEX idx_speaking_requests_show ON speaking_requests(show_id, status);

-- ============================================
-- LISTENER_SESSIONS TABLE
-- ============================================
CREATE TABLE listener_sessions (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  channel_id UUID REFERENCES channels(id) ON DELETE CASCADE,
  user_id UUID REFERENCES users(id) ON DELETE SET NULL,
  anonymous_id TEXT, -- for non-authenticated listeners
  joined_at TIMESTAMPTZ DEFAULT NOW(),
  left_at TIMESTAMPTZ,
  ip_hash TEXT, -- hashed IP for deduplication
  user_agent TEXT
);

CREATE INDEX idx_listener_sessions_channel ON listener_sessions(channel_id,
joined_at);

-- ============================================
-- ROW LEVEL SECURITY POLICIES
-- ============================================

-- Users: read own, admin read all
ALTER TABLE users ENABLE ROW LEVEL SECURITY;

CREATE POLICY "Users can read own profile"
  ON users FOR SELECT
  USING (auth.uid() = id OR EXISTS (
    SELECT 1 FROM users u WHERE u.id = auth.uid() AND u.role = 'admin'
  ));
```

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

-- Shows: public read, DJ/admin write
ALTER TABLE shows ENABLE ROW LEVEL SECURITY;

CREATE POLICY "Shows are viewable by everyone"
  ON shows FOR SELECT USING (true);

CREATE POLICY "DJs can manage own shows"
  ON shows FOR ALL
  USING (auth.uid() = host_id OR EXISTS (
    SELECT 1 FROM users u WHERE u.id = auth.uid() AND u.role IN ('admin', 'dj')
  ));

-- Schedule: public read, DJ/admin write
ALTER TABLE schedule ENABLE ROW LEVEL SECURITY;

CREATE POLICY "Schedule is viewable by everyone"
  ON schedule FOR SELECT USING (true);

CREATE POLICY "DJs can manage schedule"
  ON schedule FOR ALL
  USING (EXISTS (
    SELECT 1 FROM users u WHERE u.id = auth.uid() AND u.role IN ('admin', 'dj')
  ));

-- Tracks: public read active, uploader/admin write
ALTER TABLE tracks ENABLE ROW LEVEL SECURITY;

CREATE POLICY "Active tracks are viewable by everyone"
  ON tracks FOR SELECT USING (is_active = true);

CREATE POLICY "Admins can manage tracks"
  ON tracks FOR ALL
  USING (EXISTS (
    SELECT 1 FROM users u WHERE u.id = auth.uid() AND u.role = 'admin'
  ));

-- Playlists: public read, admin write
ALTER TABLE playlists ENABLE ROW LEVEL SECURITY;
ALTER TABLE playlist_tracks ENABLE ROW LEVEL SECURITY;

CREATE POLICY "Playlists are viewable by everyone"
  ON playlists FOR SELECT USING (true);

CREATE POLICY "Playlist tracks are viewable by everyone"
  ON playlist_tracks FOR SELECT USING (true);

-- Episodes: public read published, admin write
ALTER TABLE episodes ENABLE ROW LEVEL SECURITY;

CREATE POLICY "Published episodes are viewable by everyone"
  ON episodes FOR SELECT USING (is_published = true);

CREATE POLICY "Admins can manage episodes"
  ON episodes FOR ALL
  USING (EXISTS (
    SELECT 1 FROM users u WHERE u.id = auth.uid() AND u.role = 'admin'
  ));

-- Speaking requests: host/DJ read, user create own
ALTER TABLE speaking_requests ENABLE ROW LEVEL SECURITY;
```

```
CREATE POLICY "Hosts can view show requests"
  ON speaking_requests FOR SELECT
  USING (EXISTS (
    SELECT 1 FROM shows s WHERE s.id = show_id AND s.host_id = auth.uid()
  ) OR auth.uid() = user_id);

CREATE POLICY "Listeners can create requests"
  ON speaking_requests FOR INSERT
  WITH CHECK (auth.uid() = user_id);

-- Channels: public read, admin write
ALTER TABLE channels ENABLE ROW LEVEL SECURITY;

CREATE POLICY "Channels are viewable by everyone"
  ON channels FOR SELECT USING (true);

-- Recordings: admin only
ALTER TABLE recordings ENABLE ROW LEVEL SECURITY;

CREATE POLICY "Recordings admin only"
  ON recordings FOR ALL
  USING (EXISTS (
    SELECT 1 FROM users u WHERE u.id = auth.uid() AND u.role = 'admin'
  ));

-- Listener sessions: admin only
ALTER TABLE listener_sessions ENABLE ROW LEVEL SECURITY;

CREATE POLICY "Listener sessions admin only"
  ON listener_sessions FOR ALL
  USING (EXISTS (
    SELECT 1 FROM users u WHERE u.id = auth.uid() AND u.role = 'admin'
  ));

-- ============================================
-- REALTIME PUBLICATION
-- ============================================
ALTER PUBLICATION supabase_realtime ADD TABLE channels;
ALTER PUBLICATION supabase_realtime ADD TABLE speaking_requests;
```

API Routes (Vercel Serverless Functions)

1. Token Generation (`/api/token/route.ts`)

```
import { NextRequest, NextResponse } from 'next/server';
import { RtcTokenBuilder, RtmTokenBuilder, RtcRole, RtmRole } from 'agora-
token';

const APP_ID = process.env.NEXT_PUBLIC_AGORA_APP_ID!;
const APP_CERTIFICATE = process.env.AGORA_APP_CERTIFICATE!;

export async function POST(req: NextRequest) {
  const { channelName, uid, role = 'host' } = await req.json();

  if (!channelName || !uid) {
    return NextResponse.json({ error: 'Missing channelName or uid' }, { status:
400 });
  }
```

```
  const expirationTimeInSeconds = 3600;
  const currentTimestamp = Math.floor(Date.now() / 1000);
  const privilegeExpiredTs = currentTimestamp + expirationTimeInSeconds;

  let token;
  if (role === 'host') {
    token = RtcTokenBuilder.buildTokenWithUid(
      APP_ID,
      APP_CERTIFICATE,
      channelName,
      uid,
      RtcRole.PUBLISHER,
      privilegeExpiredTs
    );
  } else {
    token = RtcTokenBuilder.buildTokenWithUid(
      APP_ID,
      APP_CERTIFICATE,
      channelName,
      uid,
      RtcRole.SUBSCRIBER,
      privilegeExpiredTs
    );
  }

  return NextResponse.json({ token });
}
```

2. Cloud Recording - Start (`/api/recording/start/route.ts`)

```
import { NextRequest, NextResponse } from 'next/server';
import { createClient } from '@supabase/supabase-js';

const AGORA_API_BASE = 'https://api.agora.io/v1/apps';
const APP_ID = process.env.NEXT_PUBLIC_AGORA_APP_ID!;
const CUSTOMER_ID = process.env.AGORA_CUSTOMER_ID!;
const CUSTOMER_CERT = process.env.AGORA_CUSTOMER_CERTIFICATE!;
const BUCKET_NAME = process.env.AWS_BUCKET_NAME!;
const REGION = process.env.AWS_REGION!;

function getBasicAuth() {
  return 'Basic ' + Buffer.from(`${CUSTOMER_ID}:${CUSTOMER_CERT}
`).toString('base64');
}

function getAgoraRegionCode(region: string): number {
  const regionMap: Record<string, number> = {
    'us-east-1': 0,
    'us-west-1': 1,
    'eu-west-1': 2,
    'ap-southeast-1': 3,
    'ap-northeast-1': 4,
    'ap-south-1': 5,
    'sa-east-1': 6,
  };
  return regionMap[region] || 0;
}

export async function POST(req: NextRequest) {
  const { channelName, uid = 0, mode = 'mix', showId } = await req.json();

  // 1. ACQUIRE resource ID
  const acquireRes = await fetch(
```

```
    `${AGORA_API_BASE}/${APP_ID}/cloud_recording/acquire`,
    {
      method: 'POST',
      headers: {
        'Authorization': getBasicAuth(),
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({
        cname: channelName,
        uid: uid.toString(),
        clientRequest: { resourceExpiredHour: 24 }
      })
    }
  );

  if (!acquireRes.ok) {
    return NextResponse.json({ error: 'Failed to acquire resource' }, { status:
502 });
  }

  const { resourceId } = await acquireRes.json();

  // Generate token for recording
  const token = generateRecordingToken(channelName, uid);

  // 2. START recording
  const startRes = await fetch(
    `${AGORA_API_BASE}/${APP_ID}/cloud_recording/resourceid/${resourceId}/mode/$
{mode}/start`,
    {
      method: 'POST',
      headers: {
        'Authorization': getBasicAuth(),
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({
        cname: channelName,
        uid: uid.toString(),
        clientRequest: {
          token,
          recordingConfig: {
            maxIdleTime: 30,
            streamTypes: 0, // audio only
            channelType: 1, // live broadcast
            transcodingConfig: {
              height: 640,
              width: 360,
              bitrate: 500,
              fps: 15,
              mixedVideoLayout: 1,
              backgroundColor: '#000000'
            }
          },
          recordingFileConfig: {
            avFileType: ['hls', 'mp4']
          },
          storageConfig: {
            vendor: 1, // 1 = AWS S3
            region: getAgoraRegionCode(REGION),
            bucket: BUCKET_NAME,
            accessKey: process.env.AWS_ACCESS_KEY!,
            secretKey: process.env.AWS_SECRET_KEY!,
            fileNamePrefix: ['recordings', channelName]
          }
```

```
        }
      })
    }
  );

  if (!startRes.ok) {
    return NextResponse.json({ error: 'Failed to start recording' }, { status:
502 });
  }

  const { sid, serverResponse } = await startRes.json();

  // 3. Save to database
  const supabase = createClient(
    process.env.NEXT_PUBLIC_SUPABASE_URL!,
    process.env.SUPABASE_SERVICE_ROLE_KEY!
  );

  // Create episode record
  const { data: episode, error: episodeError } = await supabase
    .from('episodes')
    .insert({
      show_id: showId,
      title: `Recording ${new Date().toISOString()}`,
      mode: 'live',
      recording_resource_id: resourceId,
      recording_sid: sid,
      is_published: false
    })
    .select()
    .single();

  await supabase.from('recordings').insert({
    episode_id: episode.id,
    channel_name: channelName,
    resource_id: resourceId,
    sid,
    status: 'recording'
  });

  return NextResponse.json({ resourceId, sid, serverResponse, episodeId:
episode.id });
}
```

3. Cloud Recording - Stop (`/api/recording/stop/route.ts`)

```
import { NextRequest, NextResponse } from 'next/server';
import { createClient } from '@supabase/supabase-js';

const AGORA_API_BASE = 'https://api.agora.io/v1/apps';
const APP_ID = process.env.NEXT_PUBLIC_AGORA_APP_ID!;
const CUSTOMER_ID = process.env.AGORA_CUSTOMER_ID!;
const CUSTOMER_CERT = process.env.AGORA_CUSTOMER_CERTIFICATE!;

function getBasicAuth() {
  return 'Basic ' + Buffer.from(`${CUSTOMER_ID}:${CUSTOMER_CERT}
`).toString('base64');
}

export async function POST(req: NextRequest) {
  const { channelName, resourceId, sid, mode = 'mix', uid = 0 } = await
req.json();
```

```
  if (!channelName || !resourceId || !sid) {
    return NextResponse.json({ error: 'Missing required fields' }, { status: 400
});
  }

  const stopRes = await fetch(
    `${AGORA_API_BASE}/${APP_ID}/cloud_recording/resourceid/${resourceId}/sid/$
{sid}/mode/${mode}/stop`,
    {
      method: 'POST',
      headers: {
        'Authorization': getBasicAuth(),
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({
        cname: channelName,
        uid: uid.toString(),
        clientRequest: {}
      })
    }
  );

  if (!stopRes.ok) {
    return NextResponse.json({ error: 'Failed to stop recording' }, { status:
502 });
  }

  const data = await stopRes.json();

  // Update recording status and file list
  const supabase = createClient(
    process.env.NEXT_PUBLIC_SUPABASE_URL!,
    process.env.SUPABASE_SERVICE_ROLE_KEY!
  );

  await supabase
    .from('recordings')
    .update({
      status: 'stopped',
      stopped_at: new Date().toISOString(),
      file_list: data.serverResponse?.fileList
    })
    .eq('sid', sid);

  return NextResponse.json(data);
}
```

4. Webhook Handler with Signature Verification (`/api/webhooks/agora/route.ts`)

```
import { NextRequest, NextResponse } from 'next/server';
import { createHmac } from 'crypto';
import { createClient } from '@supabase/supabase-js';

const WEBHOOK_SECRET = process.env.AGORA_WEBHOOK_SECRET!;

function verifySignature(body: string, signature: string): boolean {
  const expected = createHmac('sha256',
WEBHOOK_SECRET).update(body).digest('hex');
  return signature === expected;
}
```

```
export async function POST(req: NextRequest) {
  const signature = req.headers.get('x-agora-signature') || '';
  const body = await req.text();

  if (!verifySignature(body, signature)) {
    return NextResponse.json({ error: 'Invalid signature' }, { status: 401 });
  }

  const event = JSON.parse(body);
  const supabase = createClient(
    process.env.NEXT_PUBLIC_SUPABASE_URL!,
    process.env.SUPABASE_SERVICE_ROLE_KEY!
  );

  switch (event.eventType) {
    case 31: // Recording file upload done
      await handleRecordingComplete(event, supabase);
      break;
    case 40: // Recorder error
      await handleRecordingError(event, supabase);
      break;
    default:
      console.log(`Unhandled event type: ${event.eventType}`);
  }

  return NextResponse.json({ received: true });
}

async function handleRecordingComplete(event: any, supabase: any) {
  const { sid, cname, fileList } = event;

  // Update recording with file URLs
  await supabase
    .from('recordings')
    .update({
      status: 'stopped',
      file_list: fileList
    })
    .eq('sid', sid);

  // Update episode with audio URL
  const { data: recording } = await supabase
    .from('recordings')
    .select('episode_id')
    .eq('sid', sid)
    .single();

  if (recording?.episode_id) {
    await supabase
      .from('episodes')
      .update({
        audio_url: fileList?.[0]?.fileName,
        is_published: true,
        published_at: new Date().toISOString()
      })
      .eq('id', recording.episode_id);
  }
}

async function handleRecordingError(event: any, supabase: any) {
  const { sid } = event;
  await supabase
    .from('recordings')
    .update({ status: 'failed' })
```

```
    .eq('sid', sid);
}
```

5. Auto-DJ Control Endpoints

```
// /api/auto-dj/start/route.ts
import { NextRequest, NextResponse } from 'next/server';

const BOT_URL = process.env.AUTO_DJ_BOT_URL!;
const BOT_SECRET = process.env.AUTO_DJ_BOT_SECRET!;

export async function POST(req: NextRequest) {
  const { channelName, playlistId } = await req.json();

  const response = await fetch(`${BOT_URL}/start`, {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      'X-Bot-Secret': BOT_SECRET
    },
    body: JSON.stringify({ channelName, playlistId })
  });

  if (!response.ok) {
    return NextResponse.json({ error: 'Failed to start Auto-DJ bot' }, { status:
502 });
  }

  const data = await response.json();
  return NextResponse.json(data);
}
```

External Auto-DJ Bot

Bot Architecture

Deploy a lightweight Node.js application on Railway/Render/EC2:

```
// bot/src/index.ts
import express from 'express';
import AgoraRTC from 'agora-rtc-sdk-ng';
import { createClient } from '@supabase/supabase-js';

const app = express();
const supabase = createClient(process.env.SUPABASE_URL!,
process.env.SUPABASE_KEY!);
const rtcClient = new AgoraRTC();

interface Track {
  id: string;
  storage_url: string;
  duration: number;
}

let currentPlaylist: Track[] = [];
let isPlaying = false;
let currentIndex = 0;

// Start Auto-DJ
```

```
app.post('/start', async (req, res) => {
  const { channelName, playlistId } = req.body;

  // Fetch playlist tracks
  const { data: tracks } = await supabase
    .from('playlist_tracks')
    .select('track_id, tracks(*)')
    .eq('playlist_id', playlistId)
    .order('sort_order');

  currentPlaylist = tracks.map(pt => pt.tracks);
  currentIndex = 0;

  // Join Agora channel
  await rtcClient.join(
    process.env.AGORA_APP_ID!,
    channelName,
    generateToken(channelName, 0),
    0
  );

  isPlaying = true;
  playNextTrack();

  res.json({ status: 'started', trackCount: currentPlaylist.length });
});

// Stop Auto-DJ
app.post('/stop', async (req, res) => {
  isPlaying = false;
  await rtcClient.leave();
  res.json({ status: 'stopped' });
});

// Get current status
app.get('/status', (req, res) => {
  res.json({
    isPlaying,
    currentTrack: currentPlaylist[currentIndex] || null,
    playlistLength: currentPlaylist.length
  });
});

function playNextTrack() {
  if (!isPlaying || currentPlaylist.length === 0) return;

  const track = currentPlaylist[currentIndex];
  // Play audio track via Agora
  // Implementation depends on Agora SDK version

  // Update "Now Playing" in Supabase
  supabase
    .from('channels')
    .update({ current_track_id: track.id })
    .eq('slug', 'main');

  // Schedule next track
  setTimeout(() => {
    currentIndex = (currentIndex + 1) % currentPlaylist.length;
    playNextTrack();
  }, track.duration * 1000);
}

app.listen(3000, () => console.log('Auto-DJ Bot running on port 3000'));
```

Bot Dependencies

```
{
  "dependencies": {
    "agora-rtc-sdk-ng": "^1.2.0",
    "express": "^4.18.2",
    "@supabase/supabase-js": "^2.49.0"
  }
}
```

Frontend Implementation

Radio Player Component

```
// components/RadioPlayer.tsx
import { useEffect, useState } from 'react';
import { useAgora } from '@agoraio-extensions/agora-rtm-react';

interface RadioPlayerProps {
  channelName: string;
  mode: 'live' | 'auto-dj';
}

export function RadioPlayer({ channelName, mode }: RadioPlayerProps) {
  const [audioSource, setAudioSource] = useState<'agora' | 'icecast'>('agora');
  const [isPlaying, setIsPlaying] = useState(false);
  const [listenerCount, setListenerCount] = useState(0);
  const [nowPlaying, setNowPlaying] = useState<{ title: string; artist: string }
| null>(null);

  // Agora RTC for live mode
  const { localAudioTrack, remoteUsers, joinChannel, leaveChannel } = useAgora({
    appId: process.env.NEXT_PUBLIC_AGORA_APP_ID!,
    channel: channelName
  });

  // Icecast for Auto-DJ mode
  const [icecastElement, setIcecastElement] = useState<HTMLAudioElement |
null>(null);

  useEffect(() => {
    if (mode === 'live') {
      setAudioSource('agora');
      joinChannel();
    } else {
      setAudioSource('icecast');
      const audio = new Audio(process.env.NEXT_PUBLIC_ICECAST_URL!);
      setIcecastElement(audio);
    }

    return () => {
      leaveChannel();
      if (icecastElement) icecastElement.pause();
    };
  }, [mode]);

  // Subscribe to listener count via Supabase Realtime
  useEffect(() => {
    const subscription = supabase
      .channel('channels')
```

```
      .on('postgres_changes',
        { event: 'UPDATE', schema: 'public', table: 'channels', filter:
`slug=eq.main` },
        (payload) => {
          setListenerCount(payload.new.listener_count);
          setNowPlaying(payload.new.current_track_id);
        }
      )
      .subscribe();

    return () => subscription.unsubscribe();
  }, []);

  // Emergency Fill: Switch to Auto-DJ if live fails
  useEffect(() => {
    let fallbackTimeout: NodeJS.Timeout;

    if (mode === 'live' && remoteUsers.length === 0) {
      fallbackTimeout = setTimeout(() => {
        // Trigger Auto-DJ fallback
        fetch('/api/auto-dj/start', {
          method: 'POST',
          body: JSON.stringify({ channelName, playlistId: 'fallback' })
        });
      }, 30000); // 30 second timeout
    }

    return () => clearTimeout(fallbackTimeout);
  }, [remoteUsers]);

  const togglePlay = () => {
    if (audioSource === 'agora') {
      // Toggle Agora audio
    } else if (icecastElement) {
      isPlaying ? icecastElement.pause() : icecastElement.play();
    }
    setIsPlaying(!isPlaying);
  };

  return (
    <div className="radio-player">
      <div className="player-controls">
        <button onClick={togglePlay}>
          {isPlaying ? 'Pause' : 'Play'}
        </button>
        <div className="now-playing">
          {nowPlaying && (
            <span>{nowPlaying.title} - {nowPlaying.artist}</span>
          )}
        </div>
        <div className="listener-count">
          👥 {listenerCount} listeners
        </div>
      </div>
    </div>
  );
}
```

Speaking Request Component

```
// components/SpeakingRequest.tsx
import { useState } from 'react';
import { useSupabase } from '@supabase/auth-helpers-react';
```

```
export function SpeakingRequest({ showId, userId }: { showId: string; userId:
string }) {
  const [message, setMessage] = useState('');
  const [status, setStatus] = useState<'idle' | 'pending' | 'approved'>('idle');
  const supabase = useSupabase();

  const requestToSpeak = async () => {
    const { data } = await supabase
      .from('speaking_requests')
      .insert({
        show_id: showId,
        user_id: userId,
        message,
        status: 'pending'
      })
      .select()
      .single();

    setStatus('pending');
  };

  // Subscribe to status updates
  useEffect(() => {
    const subscription = supabase
      .channel(`speaking_requests:${showId}`)
      .on('postgres_changes',
        { event: 'UPDATE', schema: 'public', table: 'speaking_requests', filter:
`user_id=eq.${userId}` },
        (payload) => {
          setStatus(payload.new.status);
        }
      )
      .subscribe();

    return () => subscription.unsubscribe();
  }, []);

  return (
    <div className="speaking-request">
      {status === 'idle' && (
        <div>
          <textarea
            value={message}
            onChange={(e) => setMessage(e.target.value)}
            placeholder="Why would you like to speak?"
          />
          <button onClick={requestToSpeak}>Request to Speak</button>
        </div>
      )}
      {status === 'pending' && <div>⏳ Your request is pending approval</div>}
      {status === 'approved' && <div>✅ You've been approved! Join the
call.</div>}
    </div>
  );
}
```

Environment Variables

```
# ========================================
# AGORA CONFIGURATION
```

```
# ============================================
NEXT_PUBLIC_AGORA_APP_ID=your_app_id_here
AGORA_APP_CERTIFICATE=your_app_certificate_here
AGORA_CUSTOMER_ID=your_customer_id_here
AGORA_CUSTOMER_CERTIFICATE=your_customer_certificate_here
AGORA_WEBHOOK_SECRET=your_webhook_secret_here

# ============================================
# SUPABASE CONFIGURATION
# ============================================
NEXT_PUBLIC_SUPABASE_URL=your_supabase_url_here
NEXT_PUBLIC_SUPABASE_ANON_KEY=your_supabase_anon_key_here
SUPABASE_SERVICE_ROLE_KEY=your_service_role_key_here

# ============================================
# CLOUD STORAGE (AWS S3)
# ============================================
AWS_ACCESS_KEY_ID=your_access_key_here
AWS_SECRET_ACCESS_KEY=your_secret_key_here
AWS_BUCKET_NAME=your_bucket_name_here
AWS_REGION=your_region_here

# ============================================
# AUTO-DJ BOT (External Service)
# ============================================
AUTO_DJ_BOT_URL=https://your-bot.railway.app
AUTO_DJ_BOT_SECRET=shared_secret_between_vercel_and_bot
NEXT_PUBLIC_ICECAST_URL=https://your-icecast-server/stream.mp3

# ============================================
# APPLICATION
# ============================================
NEXT_PUBLIC_APP_URL=https://your-app.vercel.app
SCHEDULE_TIMEZONE=UTC
```

Dependencies

```
{
  "dependencies": {
    "agora-rtc-react": "^2.4.0",
    "@agoraio-extensions/agora-rtm-react": "^2.2.0",
    "@supabase/supabase-js": "^2.49.0",
    "agora-token": "^2.0.3",
    "next": "14.2.0",
    "react": "^18.3.0",
    "react-dom": "^18.3.0",
    "zod": "^3.23.0",
    "@supabase/auth-helpers-react": "^0.4.0"
  },
  "devDependencies": {
    "@types/node": "^20.11.0",
    "@types/react": "^18.3.0",
    "typescript": "^5.3.0"
  }
}
```

Implementation Checklist

#	Task	Priority
1	Deploy external Auto-DJ bot on Railway/Render	Critical
2	Complete database schema with RLS policies	Critical
3	Implement Agora Cloud Recording (acquire → start → stop)	Critical
4	Add webhook signature verification	Critical
5	Set up Icecast server for Auto-DJ stream	Critical
6	Implement token generation API	High
7	Implement listener counting via backend counter	High
8	Replace RTM v1 with Signaling v2	High
9	Add Zod validation to all API routes	High
10	Define Emergency Fill logic (30s timeout → bot start)	High
11	Add index definitions for query performance	Medium
12	Implement channel state management	Medium

Emergency Fill Logic

```
// Emergency Fill Implementation
class EmergencyFill {
  private timeoutDuration = 30000; // 30 seconds
  private timer: NodeJS.Timeout | null = null;

  constructor(
    private channelName: string,
    private botUrl: string,
    private supabase: any
  ) {}

  // Call when DJ disconnects
  startTimeout() {
    this.timer = setTimeout(async () => {
      // Check if DJ reconnected
      const { data: channel } = await this.supabase
```

```
        .from('channels')
        .select('current_mode')
        .eq('slug', this.channelName)
        .single();

      if (channel.current_mode === 'live') {
        return; // DJ reconnected
      }

      // Start Auto-DJ fallback
      await fetch(`${this.botUrl}/start`, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({
          channelName: this.channelName,
          playlistId: 'fallback'
        })
      });

      await this.supabase
        .from('channels')
        .update({ current_mode: 'auto-dj' })
        .eq('slug', this.channelName);

    }, this.timeoutDuration);
  }

  cancel() {
    if (this.timer) {
      clearTimeout(this.timer);
      this.timer = null;
    }
  }
}
```

This corrected prompt addresses all critical issues identified in the verification report and provides a solid foundation for implementing a production-ready community radio platform.