Uptune: Master Product & Technical Specification (MVP)

Document Version: 2.2
Date: 10 June 2025
Project: Uptune MVP

1.0 Project Vision & Strategy

1.1 The Vision: The "Why"

The idea for Uptune was born from a simple, real-life moment: a game on a holiday that brought a group of people together through laughter, stories, and shared music. It created a moment of genuine connection that was effortless and joyful.

Our Mission: To bottle that magic. We are building a platform that uses the universal language of music as a playground for connection, helping friends, families, and eventually colleagues, discover the soundtracks to their relationships.

1.2 Core Product Strategy: Frictionless B2C to B2B

The MVP is a **100% free, consumer-focused (B2C) product** designed for maximum fun and minimum friction. The entire strategy is to build a product so joyful and easy to share that it grows organically.

- **The "Guest-First" Model:** We will remove all upfront logins. A user can start or join a game instantly with just a nickname. This approach is a direct and smart response to platform constraints (like Spotify's developer limits), turning a potential roadblock into a strategic advantage by prioritizing the user experience.
- **Product-Led Growth:** The free consumer app will act as the primary marketing and lead-generation engine for a future, premium B2B product, **"Uptune for Teams."**

1.3 Target Audience (MVP)

The core focus is on intimate social circles: friends, families, and couples looking for fun, new ways to connect.

2.0 The Core Product Experience

The experience is built around two pillars that work in harmony: on-demand private games and a recurring community event. The goal is to deliver a fun, engaging, and rewarding experience in every session.

2.1 The Game Menu (On-Demand Fun)

This is the heart of the app. Any user can instantly host a private game. The MVP

menu will feature:

- **The Mixtape:**
  - **Goal:** To collaboratively build the ultimate playlist around a fun theme.
  - **Flow:**
    1. The host picks a theme (e.g., "Our Guilty Pleasures," "Songs from the 90s," "The Perfect Road Trip") and invites friends.
    2. Each player privately submits 1-3 songs that fit the theme.
    3. **The Anonymous Reveal:** Once all submissions are in, the app displays the final, combined playlist, but with the submitter's name hidden. The group can listen to the mix and guess who chose which track.
    4. **The Host Reveal:** The host has a special "Reveal All" button. When clicked, the names of the submitters appear next to each song, sparking laughter and conversation.
  - **The Payoff:** Any player can click a single [Save This Mixtape to Spotify] button, triggering the one-time authentication to save the complete playlist to their own Spotify account.
- **Soundtrack Session:**
  - **Goal:** To create a perfectly sequenced soundtrack for a real-life event or mood.
  - **Flow:**
    1. The host defines an event (e.g., "Dinner Party," "Team Focus Session," "Family Holiday").
    2. The app uses guided, contextual questions to gather song submissions from collaborators (e.g., "Add a song for when guests are arriving," "Add a song for deep focus").
    3. **AI Sequencing:** Our backend will analyze all submissions (looking at tempo, energy, etc.) and intelligently order the tracks to match the natural flow of the event.
  - **The Payoff:** The host receives a perfectly sequenced, ready-to-play playlist that can be saved to their Spotify account with one click.
- **Desert Island Discs:**
  - **Goal:** To spark deep, personal conversations through music.
  - **Flow:**
    1. Each player is guided to choose their three essential songs: a "Head Song" (makes you think), a "Heart Song" (makes you feel), and a "Feet Song" (makes you dance).
    2. **One-by-One Reveal:** Instead of showing all results at once, the host reveals one person's island at a time. The app displays their three chosen

album covers and songs, putting a spotlight on them and encouraging the group to ask, "Why did you choose that?"
    - **The Payoff:** After everyone has shared, the app presents a [Save the 'Island Mix'] button, which creates a playlist of all the essential tracks chosen by the group, serving as a powerful souvenir of the conversation.

2.2 The Weekly Challenge (Community Pulse)

A global, themed music challenge for all users every week. This supplements the on-demand games by creating a shared "event" that keeps the app feeling fresh and alive.

3.0 User Flow & Onboarding

The user journey is optimized for immediate engagement.

3.1 Frictionless "Guest" Flow

1. **Start/Join a Game:** A user lands on the site and can either [Create a New Game] or [Enter Game Code] to join one.
2. **Enter Nickname:** A simple modal asks for only a nickname.
3. **Play the Game:** The user is immediately placed in the game lobby or active game. **No account creation is required.** The app uses temporary local storage to manage their guest session.

3.2 The "Conversion Event": Deferred Onboarding

The formal sign-up process is a rewarding, optional step at the end of a game.

1. **The Trigger:** After a collaborative game finishes, the primary Call to Action is **[Save Playlist to Spotify]**.
2. **The Action:** When a user clicks this, **only then** is the Spotify OAuth authentication flow initiated. We must request the necessary scopes: user-read-email, user-read-private, user-top-read, and playlist-modify-public.
3. **The Payoff:** Upon successful authorization, the app instantly creates the playlist in the user's Spotify account. It also converts their temporary "guest" data into a permanent user account in our database.

4.0 Technical Specification

4.1 Tech Stack & Architecture

- **Platform:** A responsive, mobile-first Web Application. No native iOS/Android apps for the MVP.
- **Development Environment:** Replit.

- **Backend & Database:** Google Firebase with a PostgreSQL database.
  - **Authentication:** Firebase Authentication (for when a user converts from a guest).
  - **Database:** Replit's native PostgreSQL Database.
  - **Backend Logic:** Server-side code running within the Replit environment, ready for migration to Cloud Functions.

4.2 Spotify API Integration Details

This section outlines the specific API interactions required. All sensitive calls involving the Client Secret must be handled on the backend.

1. **Getting Song IDs (Guest Experience):**
   - **Endpoint:** GET /v1/search
   - **Authorization:** This call can be made using the **Client Credentials Flow**. The backend server will use our app's Client ID and Client Secret to obtain an app-level access token. This does not require a user to be logged in.
   - **Flow:**
     1. A guest user types a search query (e.g., "Queen") in the app's UI.
     2. The frontend sends this query to our backend.
     3. The backend makes an authenticated request to the Spotify /search endpoint with type=track.
     4. The backend receives the search results and returns a formatted list to the frontend, including track_name, artist_name, album_art_url, and the crucial spotify_track_id.
   - **Result:** The Uptune app now has the necessary spotify_track_id for every song a guest selects during a game.
2. **Creating the Playlist (The "Magic Link" / Conversion Event):**
   - **Endpoint(s):** POST /v1/users/{user_id}/playlists, POST /v1/playlists/{playlist_id}/tracks
   - **Authorization:** This process uses the **Authorization Code Flow** and requires user interaction.
   - **Flow:**
     1. A user clicks the [Save Playlist to Spotify] button.
     2. The frontend redirects the user to the **Spotify Authorization URL**, which we construct. This URL must include our client_id, the redirect_uri, and the scope parameter set to playlist-modify-public. This redirect is the "magic link."
     3. The user logs into Spotify (if they aren't already) and agrees to grant Uptune permission.
     4. Spotify redirects the user back to our specified redirect_uri, providing a

one-time authorization_code in the URL parameters.

5. Our backend receives this code. It then makes a secure, server-to-server request to https://accounts.spotify.com/api/token, exchanging the code, our client_id, and our client_secret for a user-specific access_token and refresh_token.
6. With the new access_token, our backend first calls POST /v1/users/{user_id}/playlists to create a new, empty playlist in the user's account (e.g., named "Our 90s Mixtape").
7. Spotify returns the playlist_id of the new playlist.
8. Our backend then makes a final call to POST /v1/playlists/{playlist_id}/tracks, providing the list of spotify_track_ids collected during the game to populate the playlist.

4.3 PostgreSQL Database Schema

The schema must support the "guest-first" model.

- **users Table:**
  - id (SERIAL PRIMARY KEY)
  - firebase_uid (VARCHAR, UNIQUE, NULLABLE) - Populated only after they convert from guest.
  - name (VARCHAR)
  - email (VARCHAR, UNIQUE, NULLABLE)
  - profile_picture_url (TEXT)
  - spotify_refresh_token_encrypted (TEXT, NULLABLE)
  - onboarding_profile (JSONB) - For storing Desert Island Discs, etc., after they complete the post-conversion profile setup.
  - created_at (TIMESTAMPTZ)
- **parties Table:**
  - id (SERIAL PRIMARY KEY)
  - host_user_id (INTEGER REFERENCES users(id), NULLABLE)
  - host_guest_session_id (TEXT, NULLABLE)
  - invite_code (VARCHAR, UNIQUE)
  - game_type (VARCHAR) - e.g., 'TheMixtape', 'SoundtrackSession'
  - game_status (VARCHAR) - e.g., 'lobby', 'active', 'complete'
  - game_data (JSONB)
  - result_playlist_url (TEXT)
  - created_at (TIMESTAMPTZ)
- **party_members Table:**
  - party_id (INTEGER REFERENCES parties(id))
  - user_id (INTEGER REFERENCES users(id), NULLABLE)

    - guest_session_id (TEXT, NULLABLE)
    - guest_nickname (VARCHAR)
    - PRIMARY KEY (party_id, user_id, guest_session_id)
- **submissions Table:**
    - id (SERIAL PRIMARY KEY)
    - user_id (INTEGER REFERENCES users(id), NULLABLE)
    - guest_session_id (TEXT, NULLABLE)
    - parent_type (VARCHAR) - 'weeklyChallenge' or 'party'
    - parent_id (VARCHAR)
    - spotify_track_id (VARCHAR)
    - track_name, artist_name, album_art_url (TEXT)
    - created_at (TIMESTAMPTZ)
- **challenges Table:**
    - **Purpose:** Defines the active Weekly Challenge. This table will likely be managed manually in the database console at first.
    - id (VARCHAR(50) PRIMARY KEY) - A unique identifier for the week, e.g., '2025-W24'.
    - title (VARCHAR(255) NOT NULL)
    - description (TEXT)
    - is_active (BOOLEAN NOT NULL DEFAULT FALSE)
    - end_date (TIMESTAMPTZ NOT NULL)
- **votes Table:**
    - **Purpose:** Tracks votes on submissions to prevent a user from voting multiple times on the same song. This provides a scalable way to manage likes/votes.
    - submission_id (INTEGER NOT NULL REFERENCES submissions(id) ON DELETE CASCADE)
    - user_id (INTEGER REFERENCES users(id), NULLABLE)
    - guest_session_id (TEXT, NULLABLE)
    - voted_at (TIMESTAMPTZ DEFAULT NOW())
    - PRIMARY KEY (submission_id, user_id, guest_session_id)