Simplified DJAMMS Architecture Proposal (FINAL REVISION)

Core Principles

- •Maintain multi-window real-time sync as the primary requirement
- •Reduce abstraction layers from 4+ (stores → services → database) to 2 (stores → database)
- •Simplify venue_id logic by using Appwrite user ID directly
- •Consolidate state management into fewer, more focused stores
- •Add commercial-grade robustness: automatic recovery, error boundaries, offline support, and redundant sync mechanisms

---

APPWRITE FUNCTIONS

Below is a revised, consolidated list of five essential Appwrite functions, along with their detailed responsibilities.

1. Auth & Setup Handler

This function combines user authentication handling with the initial setup of a new venue. It's the core of user and venue creation.

- **Triggers:** Appwrite users.onCreate event (for new users) and a manual API call from the UI (for login).
- **Responsibilities:**
  - **User Provisioning:** On users.onCreate, it creates the user's profile in the users collection, sets their role, and initializes their preferences.
  - **Venue Creation:** If the user is a new venue owner, this function creates a new entry in the venues collection and populates it with default, empty state for now_playing, active_queue, and player_settings.
  - **Post-Login Actions:** After a user logs in, this function can retrieve their associated venue_id and preferences, preparing the UI for the session.

2. Player & Venue State Manager

This is the most critical function. It's the sole endpoint for all real-time updates from the video player. This consolidation eliminates the need for a separate Player-Instance-Manager.

- **Triggers:** A manual API call from the player (via fetch requests) on every significant event, such as a state change, a heartbeat, or an API operation acknowledgment.
- **Responsibilities:**
  - **State Updates:** Processes player-initiated events (onStateChange, onReady, onEnded, etc.). It updates the now_playing and

player_settings objects within the venues collection.

- **Heartbeat Handling:** Updates the last_heartbeat_at timestamp in the venues collection. If a new player connects, it creates a new entry in player_instances.
- **Queue Management:** Automatically manages the active_queue and priority_queue based on player events (e.g., when a video ends, it pulls the next video from the queue).

3. Playlist & Content Manager

This function centralizes all actions related to playlists and content. It's the backend for your content library and playlist management UI.

- **Triggers:** Manual API calls from the UI Admin Console for various actions.
- **Responsibilities:**
  - **Playlist Operations:** Handles creating, updating, deleting, and sharing playlists. This includes the Import-Playlist functionality you mentioned.
  - **Track Management:** Adds and removes tracks from playlists and manages the queue order.
  - **Content Gallery:** Manages the storage and metadata for uploaded content (images, videos, etc.) in a dedicated bucket.

4. UI Command & Sync Hub

This function serves as the bridge between the UI Admin Console and the player state. It's designed to handle user actions and push updates to all clients. This consolidates the UI-Instance-Manager.

- **Triggers:** A manual API call from the UI Admin Console (e.g., when a user clicks a button).
- **Responsibilities:**
  - **Command Processing:** Receives UI commands like "play," "pause," "skip," "mute," or "shuffle." It then updates the venues collection accordingly.
  - **Client Synchronization:** After processing a command, it uses Appwrite's real-time capabilities to broadcast the state change to all connected clients (both players and UIs). This ensures the UI is always in sync with the player's true state without constant polling.

5. Scheduler & Maintenance Agent

This is a scheduled, cron-like function that handles all background and periodic tasks, ensuring the system remains healthy and up-to-date.

- **Triggers:** A scheduled Appwrite cron job (e.g., every 5 minutes).
- **Responsibilities:**

- **Connection Auditing:** Iterates through all venues to check for player heartbeats. If a player hasn't sent a heartbeat within the threshold, it flags the player as disconnected.
- **Scheduled Events:** Processes any scheduled playlist or content changes from the venues collection.
- **Cleanup:** Deletes old log entries or purges outdated data to maintain database health.
- **System Notifications:** Can send alerts or logs for system-level issues to a designated channel.

DJAMMS Consolidated Database Schema

This revised schema provides a robust, logical, and maintainable foundation for all the platform's functionality.

---

1. venues

Purpose: The central hub for all venue-specific player state, configuration, and active queues.

Attributes:

- **venue_id**: **string (required)** - Unique identifier.
- **venue_name**: **string**
- **owner_id**: **string (required)** - Reference to users.user_id.
- **active_player_instance_id**: **string** - Current active player instance ID.
- **now_playing**: **JSON object** - The single source of truth for the current track.
    - video_id: **string**
    - title: **string**
    - artist: **string**
    - duration: **integer** (seconds)
    - thumbnail: **string**
    - start_timestamp: **datetime** - When playback started.
    - state: **string** - e.g., "playing", "paused", "buffering".

    - vote_count: **integer**
    - is_liked: **boolean**
- **active_queue**: **JSON Array** - The main playback queue.
    - video_id: **string**
    - title: **string**
    - artist: **string**
    - duration: **integer**
    - thumbnail: **string**
    - added_by_user_id: **string**
    - added_at: **datetime**
    - priority: **integer** - Queue position priority.
- **priority_queue**: **JSON Array** - High-priority "play next" items. **(Schema is identical to active_queue)**
    - video_id: **string**
    - title: **string**
    - artist: **string**
    - duration: **integer**
    - thumbnail: **string**
    - added_by_user_id: **string**
    - added_at: **datetime**
    - priority: **integer**
- **player_settings**: **JSON object** - All player configuration settings.
    - repeat_mode: **string** (none|one|all)
    - shuffle_enabled: **boolean**
    - shuffle_seed: **integer**
    - crossfade_time: **integer** (seconds)
    - master_volume: **integer** (0-100)

        - is_muted: **boolean**
        - eq_settings: **JSON object**
        - mic_volume: **integer** (0-100)
        - dynamic_compressor_enabled: **boolean**
        - player_size: **JSON object** (width, height)
        - player_position: **JSON object** (x, y)
        - is_fullscreen: **boolean**
        - display_sliders: **JSON object** (brightness, contrast, etc.)
        - app_name: **string**
- **schedule_data**: **JSON object** - Stores all scheduled events for the venue.
- **last_heartbeat_at**: **datetime**
- **last_updated**: **datetime**
- **created_at**: **datetime**
- **Indexes:**
    - Primary: venue_id (unique)
    - Foreign: owner_id → users.user_id
    - Performance: last_updated

---

2. users

Purpose: User profiles, preferences, and authentication.

Attributes:

- **user_id**: **string (required)**
- **email**: **string (required)**
- **username**: **string**
- **venue_id**: **string**

- **role**: **string** (admin|moderator|user)
- **preferences**: **JSON object** - User-specific settings.
    - theme: **string**
    - notifications_enabled: **boolean**
    - default_volume: **integer**
    - auto_play: **boolean**
    - language: **string**
    - timezone: **string**
    - min_to_tray_enabled: **boolean**
    - update_checks_enabled: **boolean**
    - telemetry_enabled: **boolean**
- **avatar_url**: **string**
- **is_active**: **boolean**
- **is_developer**: **boolean**
- **created_at**: **datetime**
- **last_login_at**: **datetime**
- **last_activity_at**: **datetime**
- **Indexes:**
    - Primary: user_id (unique)
    - Unique: email
    - Foreign: venue_id → venues.venue_id

---

3. playlists

Purpose: User-created and system-managed playlists.

Attributes:

- **playlist_id**: **string (required)**
- **name**: **string**
- **description**: **string**
- **owner_id**: **string**
- **venue_id**: **string**
- **is_public**: **boolean**
- **is_default**: **boolean**
- **is_starred**: **boolean** - Tracks if a user has starred it.
- **category**: **string**
- **cover_image_url**: **string**
- **tracks**: **JSON Array** - The ordered list of tracks.
    - video_id: **string**
    - title: **string**
    - artist: **string**
    - duration: **integer**
    - thumbnail: **string**
    - order: **integer**
- **track_count**: **integer**
- **total_duration**: **integer**
- **tags**: **JSON Array**
- **play_count**: **integer**
- **last_played_at**: **datetime**
- **created_at**: **datetime**
- **updated_at**: **datetime**
- **Indexes:**
    - Primary: playlist_id (unique)

- Foreign: owner_id → users.user_id, venue_id → venues.venue_id
- Performance: is_public, is_default, category

---

4. activity_log

Purpose: An immutable audit log of all system activities.

Attributes:

- **log_id**: **string (required)**
- **user_id**: **string**
- **venue_id**: **string**
- **event_type**: **string** - e.g., "playback_started", "track_added".
- **event_data**: **JSON object** - Contextual information for the event.
- **timestamp**: **datetime (required)**
- **ip_address**: **string**
- **user_agent**: **string**
- **session_id**: **string**
- **Indexes:**
    - Primary: log_id (unique)
    - Performance: user_id, venue_id, event_type, timestamp

Comprehensive DJAMMS I/O & Event Management Master List

Below is a complete, item-by-item table detailing all I/O and event-driven state management for the DJAMMS project. This list serves as the definitive reference for all communication between the Video Player, UI Admin Console, and the backend server functions. It is structured to provide a clear and actionable blueprint for developers.

---

1. Player-Initiated Events

These actions originate from the YouTube iFrame API player and inform the server of its current state. All requests from the player are handled by the **Player & Venue State Manager** function.

2. UI-Initiated Command

These actions are initiated by a user interacting with the Admin Console. They are processed by the **UI Command & Sync Hub** function. The server then pushes real-time updates to all clients via WebSocket/SSE.

3. Server-Initiated Events

These are real-time updates pushed by the server to all connected clients (players and UIs) to maintain a synchronized state. All these are handled by the server's real-time capabilities triggered by other functions.

4. Player-Specific UI Components

These components are display-only and receive their data from the NOW_PLAYING_UPDATE message. They do not initiate a request but react to a server push.

DJAMMS I/O & Event Management: Consolidated Master Table (Revised)

Type	Action Type	Action Purpose	Event/Trigger	Sender	Receiver	Request / Response Data	Associated Function
Player-Initiated Events	**Player State Update**	Notifies server of player state changes.	onStateChange (YT.PlayerState)	Video Player	Server	**Request: POST /api/player/status body: { venueId, state: 'playing', currentTime, duration }**	Player & Venue State Manager
Player-Initiated Events	**Player Initialized**	Confirms player is loaded and ready.	onReady	Video Player	Server	**Request: POST /api/player/ready body: { venueId, playerId }**	Player & Venue State Manager
Player-Initiated Events	**Player Heartbeat**	Maintains an active connection and syncs time.	setInterval (e.g., every 5s)	Video Player	Server	**Request: POST /api/player/heartbeat body: { venueId, currentTime }**	Player & Venue State Manager
Player-Initiated Events	**Player Error**	Reports a player-related error.	onError	Video Player	Server	**Request: POST /api/player/error body: { venueId, errorCode }**	Player & Venue State Manager
UI-Initiated Commands	**Playback Control**	Toggles playback state of the player.	Play/Pause Button click	UI Admin Console	Server	**Request: POST /api/ui/command body: { venueId, command: 'PLAY_PAUSE' }**	UI Command & Sync Hub
UI-Initiated Commands	**Track Navigation**	Skips to the next or previous track.	Next/Previous Button click	UI Admin Console	Server	**Request: POST /api/ui/command body: { venueId, command: 'PLAY_NEXT' }**	UI Command & Sync Hub
UI-Initiated Commands	**Volume Control**	Adjusts the player's master volume.	Master Volume Slider change	UI Admin Console	Server	**Request: POST /api/ui/command body: { venueId, command: 'SET_VOLUME', value: N }**	UI Command & Sync Hub
UI-Initiated Commands	**Crossfade Time**	Updates the crossfade duration setting.	Crossfade Time Slider change	UI Admin Console	Server	**Request: POST /api/settings/update body: { venueId, setting: 'crossfade_time', value: N }**	UI Command & Sync Hub
UI-Initiated Commands	**Shuffle Toggling**	Toggles shuffle mode for the queue.	Shuffle Button click	UI Admin Console	Server	**Request: POST /api/ui/command body: { venueId, command: 'SHUFFLE_TOGGLE' }**	UI Command & Sync Hub
UI-Initiated Commands	**Repeat Toggling**	Toggles repeat mode for the queue.	Repeat Button click	UI Admin Console	Server	**Request: POST /api/ui/command body: { venueId, command: 'REPEAT_TOGGLE' }**	UI Command & Sync Hub
UI-Initiated Commands	**Emergency Stop**	Immediately halts all audio output.	Emergency Stop Button click	UI Admin Console	Server	**Request: POST /api/ui/command body: { venueId, command: 'EMERGENCY_STOP' }**	UI Command & Sync Hub
UI-Initiated Commands	**Fade Out Stop**	Gradually fades out audio playback.	Fade Out Stop Button click	UI Admin Console	Server	**Request: POST /api/ui/command body: { venueId, command: 'FADE_OUT_STOP' }**	UI Command & Sync Hub
UI-Initiated Commands	**Add to Queue**	Adds a track to the active queue.	Add to Queue Button click	UI Admin Console	Server	**Request: POST /api/queue/add body: { venueId, videoId, position: 'end' }**	Playlist & Content Manager
UI-Initiated Commands	**Remove from Queue**	Removes a track from the queue.	Remove from Queue Button click	UI Admin Console	Server	**Request: POST /api/queue/remove body: { venueId, videoId }**	Playlist & Content Manager
UI-Initiated Commands	**Quick Add**	Adds a predefined track to the queue.	Quick Add Buttons click	UI Admin Console	Server	**Request: POST /api/queue/add body: { venueId, videoId, action: 'quick' }**	Playlist & Content Manager
UI-Initiated Commands	**Search Query**	Retrieves search results from the library.	Search Input Field input	UI Admin Console	Server	**Request: GET /api/search?q=query&filters=...**	Playlist & Content Manager
UI-Initiated Commands	**Track Preview**	Requests a preview URL for a track.	Track Preview Button click	UI Admin Console	Server	**Request: GET /api/preview?videoId=...**	Playlist & Content Manager
UI-Initiated Commands	**Like/Vote**	Records a user's vote on a track.	Like/Vote Button click	UI Admin Console	Server	**Request: POST /api/track/vote body: { venueId, videoId, voteType: 'like' }**	UI Command & Sync Hub
UI-Initiated Commands	**Content Upload**	Uploads new media to the content gallery.	Content Upload Form submit	UI Admin Console	Server	**Request: POST /api/content/upload body: { file, venueId }**	Playlist & Content Manager
UI-Initiated Commands	**Delete Content**	Deletes a media item from the gallery.	Delete Selected Button click	UI Admin Console	Server	**Request: POST /api/content/delete body: { venueId, contentIds: [...] }**	Playlist & Content Manager
UI-Initiated Commands	**Playlist Management**	Creates, stars, or modifies a playlist.	Playlist Cards, Star/Unstar Toggle click	UI Admin Console	Server	**Request: POST /api/playlist/manage body: { action: 'star', playlistId, userId }**	Playlist & Content Manager
UI-Initiated Commands	**Schedule Management**	Creates or modifies a schedule entry.	Calendar Grid click, form changes	UI Admin Console	Server	**Request: POST /api/schedule/update body: { venueId, scheduleData: { ... } }**	UI Command & Sync Hub
UI-Initiated Commands	**System Settings**	Modifies player or system settings.	EQ Controls, Network Settings change	UI Admin Console	Server	**Request: POST /api/settings/update body: { venueId, setting: 'eq', value: { ... } }**	UI Command & Sync Hub
UI-Initiated Commands	**User Preferences**	Updates user-specific settings.	User Preferences change	UI Admin Console	Server	**Request: POST /api/user/preferences body: { userId, preferences: { ... } }**	Auth & Setup Handler
UI-Initiated Commands	**Authentication**	Enables/disables auth settings or manages keys.	Authentication Toggles, API Key Management click	UI Admin Console	Server	**Request: POST /api/auth/settings body: { setting: '2fa_enabled', value: true }**	Auth & Setup Handler
UI-Initiated Commands	**Backup/Restore**	Triggers a system backup or restore.	Backup/Restore Buttons click	UI Admin Console	Server	**Request: POST /api/system/backup body: { venueId }**	UI Command & Sync Hub
UI-Initiated Commands	**Download Log**	Triggers the download of log files.	Log Download Button click	UI Admin Console	Server	**Request: GET /api/logs/download body: { venueId }**	Scheduler & Maintenance Agent
Server-Initiated Events	**Now Playing Update**	Syncs all clients with the current track info.	Player state change or initial UI connection	Server	UI Admin Console & Player	**Response: WebSocket/SSE message: { type: 'NOW_PLAYING_UPDATE', payload: { title, artist, duration, thumbnail, state, start_timestamp } }**	UI Command & Sync Hub
Server-Initiated Events	**Player Disconnected**	Notifies UI that the player is offline.	Heartbeat timeout on the server	Server	UI Admin Console	**Response: WebSocket/SSE message: { type: 'PLAYER_DISCONNECTED', venueId }**	Scheduler & Maintenance Agent
Server-Initiated Events	**Queue Update**	Syncs all clients with the latest queue state.	Add/Remove from Queue request	Server	UI Admin Console	**Response: WebSocket/SSE message: { type: 'QUEUE_UPDATE', payload: { active_queue, priority_queue } }**	UI Command & Sync Hub
Server-Initiated Events	**System Status**	Notifies of system-wide changes.	Scheduled task or system control request	Server	UI Admin Console	**Response: WebSocket/SSE message: { type: 'SYSTEM_STATUS', status: 'backup_in_progress' }**	Scheduler & Maintenance Agent
Server-Initiated Events	**Log Update**	Streams real-time log entries to the UI.	New log entry in activity_log	Server	UI Admin Console	**Response: WebSocket/SSE message: { type: 'LOG_ENTRY', entry: { ... } }**	All Functions
Server-Initiated Events	**User Authentication**	Pushes user data and permissions on login.	Successful login event	Server	UI Admin Console	**Response: WebSocket/SSE message: { type: 'AUTH_SUCCESS', payload: { userId, role, preferences } }**	Auth & Setup Handler
Server-Initiated Events	**Search Results Update**	Pushes search results to the UI.	Search query request completed	Server	UI Admin Console	**Response: WebSocket/SSE message: { type: 'SEARCH_RESULTS_UPDATE', payload: [...] }**	Playlist & Content Manager
Server-Initiated Events	**Content Gallery Update**	Pushes an updated content list.	Upload/delete request completed	Server	UI Admin Console	**Response: WebSocket/SSE message: { type: 'CONTENT_GALLERY_UPDATE', payload: [...] }**	Playlist & Content Manager
Server-Initiated Events	**Playlist Content Update**	Pushes the content of a selected playlist.	Playlist card click	Server	UI Admin Console	**Response: WebSocket/SSE message: { type: 'PLAYLIST_CONTENT_UPDATE', payload: { playlist } }**	Playlist & Content Manager
Player-Specific UI Components	**Video Thumbnail**	Displays current track thumbnail.	NOW_PLAYING_UPDATE event	Server	UI Admin Console	payload.thumbnail	N/A
Player-Specific UI Components	**Video Title**	Displays current track title.	NOW_PLAYING_UPDATE event	Server	UI Admin Console	payload.title	N/A
Player-Specific UI Components	**Video Metadata**	Displays channel and duration.	NOW_PLAYING_UPDATE event	Server	UI Admin Console	payload.channel, payload.duration	N/A
Player-Specific UI Components	**Progress Bar**	Visualizes playback progress.	NOW_PLAYING_UPDATE event	Server	UI Admin Console	payload.start_timestamp, payload.duration	N/A
Player-Specific UI Components	**Queue List**	Displays the list of upcoming songs.	QUEUE_UPDATE event	Server	UI Admin Console	payload.active_queue	N/A
Player-Specific UI Components	**Log Display Panel**	Displays real-time logs.	LOG_ENTRY_UPDATE event	Server	UI Admin Console	payload.entry	N/A
Player-Specific UI Components	**Search Results Display**	Displays a list of tracks.	SEARCH_RESULTS_UPDATE event	Server	UI Admin Console	payload.results	N/A
Player-Specific UI Components	**Playlist Cards**	Displays playlists.	PLAYLIST_CONTENT_UPDATE event	Server	UI Admin Console	payload.playlists	N/A
Player-Specific UI Components	**Content Gallery**	Displays uploaded media items.	CONTENT_GALLERY_UPDATE event	Server	UI Admin Console	payload.items	N/A
Player-Specific UI Components	**Calendar Grid**	Displays schedule slots.	SCHEDULE_DATA_UPDATE event	Server	UI Admin Console	payload.schedule	N/A