BandhuConnect+ Backend Guidelines: Hackathon MVP (Descriptive) 🚀

This guideline provides a focused and detailed approach to building the backend for your BandhuConnect+ MVP within a hackathon timeframe. It prioritizes core functionality, rapid development, and essential security, explicitly leveraging your chosen Supabase stack. This documentation is structured to be clear and actionable, even for an AI app builder, explaining the *why* and *how* behind each recommendation.

1. Architectural Principles (MVP Focus)

For the hackathon, we'll adhere to these core principles. The goal isn't a hyper-optimized production system, but a **demonstrably functional prototype** that showcases the main features. We'll rely heavily on Supabase's managed nature to abstract away complex infrastructure concerns.

- **Functionality:** The **primary objective** is to ensure that the core user journeys (e.g., volunteer sign-up, task display, basic chat, pilgrim requesting help, admin viewing requests) are operational and can be demonstrated. This means prioritizing working features over intricate edge-case handling.
- **Rapid Development:** We will choose the **quickest and most direct path to implementation**. This often means leveraging Supabase's client-side SDKs and pre-built features extensively, minimizing custom server-side code unless absolutely necessary for a core demo.
- **Basic Scalability & Reliability:** By using **Supabase's built-in scaling and uptime**, we automatically gain a level of system resilience suitable for an MVP. We won't need to implement custom load balancing or failover strategies during the hackathon. Supabase handles the underlying PostgreSQL database and authentication services in a robust, managed way.
- **Essential Security:** Our focus will be on implementing **authentication** (knowing *who* the user is) and **basic data access control** (ensuring users can only interact with data they're supposed to). This mainly involves configuring Supabase's Row Level Security (RLS) and authentication features.

2. Tech Stack & Services (Leveraging Supabase for MVP)

Supabase offers many features out-of-the-box that are perfect for rapid hackathon development, significantly reducing the need to write custom backend code.

2.1. Database (PostgreSQL via Supabase)

Supabase provides a powerful PostgreSQL database. We'll define its structure and access rules.

- **Schema Design:** This is the blueprint for our data. We will set up the following tables:
    - users: Stores information for all participants (volunteers, admins, pilgrims).
    - requests: Holds all incoming help requests from pilgrims/attendees.
    - volunteer_tasks: Links specific requests to assigned volunteers and tracks task status.
    - messages: Stores chat messages for the general chatroom.
    - **Focus:** It's crucial to define **Primary Keys (PKs)** (unique identifiers for each record, typically uuid type) and **Foreign Keys (FKs)** (links between tables, like user_id in requests linking to users.id) correctly. This ensures data integrity and allows efficient querying.
    - **Data Types:** Use appropriate PostgreSQL data types. For example, uuid for IDs, text for names, emails, descriptions, enum for predefined statuses (like 'pending', 'assigned'), float8 for latitude/longitude, or the more precise geography(point) type for spatial data, which is ideal for location tracking.
- **Basic Indexing:** For efficient data retrieval, we need indexes.
    - **Purpose:** Indexes are like an alphabetical list for a book; they speed up searches. We'll ensure indexes are applied to all **Foreign Key (FK)** columns (as these are frequently used for joining tables) and any columns used in common WHERE clauses (e.g., created_at for ordering, status for filtering). Supabase often creates some essential indexes automatically.
- **Row Level Security (RLS):** This is an **absolutely critical security layer** for Supabase, even for an MVP.
    - **Purpose:** RLS allows us to define fine-grained access policies directly at the database level. This means a user's database query will only return rows they are authorized to see, regardless of what the application code requests.
    - **Implementation:** We will implement basic RLS policies to demonstrate that:
        - **User Profiles:** Users can only see/edit their *own* user profile data.
        - **Pilgrim Requests:** Pilgrims can only see the requests *they* have submitted.
        - **Volunteer Tasks:** Volunteers can only see tasks that have been explicitly *assigned to them*.

        - **Admin Access:** Administrators, identified by their role, can see and modify all relevant data across users, requests, and volunteer_tasks tables.

- **Transactions:** While full transactional integrity (ACID properties) might be a post-hackathon feature, for critical operations like assigning a task that involves updating a request's status and creating a volunteer_task entry, an AI builder should be aware that these operations ideally require atomic execution. For the hackathon, a simple sequential update might suffice, but acknowledge that a transaction is the correct approach.

2.2. Authentication (Supabase Auth)

Supabase Auth provides a managed authentication service, simplifying user management.

- **User Registration & Login:** We will implement user sign-up and login functionality for all three user roles (volunteer, admin, pilgrim).
    - **Methods:** Leverage Supabase Auth's capabilities for Phone OTP (One-Time Password) or Email/Password-based authentication. This makes it easy to add new users.
- **Role-Based Access Control (RBAC):** Users will have an assigned role.
    - **Implementation:** Use a role column (e.g., an enum type: 'volunteer', 'admin', 'pilgrim') in your users table. This role can then be used in your **RLS policies** and for simple **client-side UI rendering logic** to differentiate permissions (e.g., an admin sees different dashboard components than a volunteer).
- **Secure Token Handling:** Supabase Auth handles **JSON Web Tokens (JWTs)** automatically.
    - **Purpose:** JWTs are secure tokens that verify a user's identity and permissions. Supabase manages their creation, refreshing, and secure storage on the client-side (e.g., in localStorage or secure storage in React Native), abstracting away complex token management for the developer.

2.3. Real-time (Supabase Realtime)

Supabase Realtime enables instant updates across connected clients.

- **Core Real-time Flows:** This is crucial for the dynamic nature of BandhuConnect+. We will implement Realtime for the most impactful user experiences:

    - **Request Flow:** As soon as a pilgrim creates a new request, administrators (and potentially nearby available volunteers, though that's a stretch) will see this new request appear on their dashboard **immediately** without manual refreshing.
    - **Assignment Flow:** When an admin assigns a task to a volunteer, the volunteer's mobile app dashboard will **instantly update** to display their newly assigned task.
    - **Chat Flow:** Messages sent in the general chatroom will appear for all participants in that channel **in real-time**.
- **Channel Design:** For the MVP, we'll start with simple real-time channels.
    - **Examples:** One general chat channel (chat_general), and a channel for admin request updates (admin_requests) could be sufficient. As the app grows, more granular channels (e.g., chat_skill_med) would be added.

2.4. File Storage (Supabase Storage)

Supabase Storage provides a robust solution for managing user-uploaded files.

- **Photo Uploads:** We will implement the functionality for pilgrims to upload photos for two critical features:
    - "Lost Person" reports (e.g., a photo of the missing person).
    - "Sanitation Issues" reports (e.g., a photo of the unsanitary area as proof).
- **Storage Buckets & Policies:**
    - **Buckets:** Create at least one dedicated storage bucket (e.g., user-uploads) within Supabase. For production, more granular buckets like lost-person-photos and sanitation-proofs would be better.
    - **Policies:** Set up basic **storage policies** to ensure:
        - Authenticated users can upload files to the designated bucket(s).
        - Viewing policies ensure that only relevant users (e.g., admins, assigned volunteers for a task) can access specific sensitive files (like lost person photos).

2.5. Backend Logic (Supabase Edge Functions / Basic Server Logic)

For operations requiring server-side logic beyond direct database access, we'll use Supabase Edge Functions (serverless functions).

- **Task Assignment (Simplified):** The "algorithm" for assigning tasks will be very basic for the MVP.
  - **Implementation:** This could initially be implemented with a simple function (either directly in your React Native client for the hackathon, or as a very basic Supabase Edge Function if time permits) that performs a direct assignment. For example, it could assign a hardcoded task to the *first available volunteer* that has a 'general' skill, or is within a simulated proximity.
  - **Focus:** The goal here is to **demonstrate *a task being assigned*** and reflected on the volunteer's dashboard, not to build a sophisticated optimization engine.
- **Notifications (Basic Triggers):** Set up simple server-side logic (e.g., within an Edge Function or a database trigger if Supabase supports it for this use case) that sends a basic notification.
  - **Example:** When a volunteer_tasks record is inserted (meaning a task is assigned), a notification can be sent to the relevant volunteer's device.
- **Data Validation:** Implement **basic server-side validation** for critical fields (e.g., ensuring required fields are present, basic type checks) within your Edge Functions or database CHECK constraints. For less critical user input, rely on client-side validation to save development time during the hackathon.

3. API Design (MVP Focus)

For the hackathon, our API interaction will be direct and simplified.

- **Direct Supabase Client:** You will primarily interact with Supabase directly from your React Native app. The **Supabase client libraries** (e.g., @supabase/supabase-js) abstract away the raw HTTP API calls, making it very efficient for CRUD (Create, Read, Update, Delete) operations.
- **Resource Interaction:** Focus on using the Supabase client to perform standard database operations against your users, requests, volunteer_tasks, and messages tables.
- **Error Handling:** Leverage the error handling provided by the Supabase client. For the UI, for presentation purposes, show a simple alert or a message on the screen for any API errors encountered. Full-fledged global error handling and detailed user-friendly error messages can be refined post-hackathon.

4. Security Considerations (MVP Essentials)

Even for an MVP, core security is non-negotiable.

- **RLS (Crucial):** As emphasized, correctly configuring **Row Level Security policies** in PostgreSQL via the Supabase UI is the most important security measure to prevent unauthorized data access.
- **Environment Variables: Never hardcode sensitive information** like your Supabase URL or anon key directly into your codebase. Store them securely using React Native's environment variable capabilities (e.g., .env files and tools like react-native-dotenv or expo-constants).
- **HTTPS:** Supabase automatically handles **HTTPS** for all API communication. This encrypts data in transit, protecting it from eavesdropping.
- **Auth Flow:** Ensure that your chosen authentication flow (Phone OTP or Email/Password) is correctly implemented and effectively secures user sessions, allowing only authenticated and authorized users to interact with the app's features.

5. Deployment & Operations (Hackathon Level)

These steps focus on getting your backend running and debuggable for the hackathon.

- **Supabase Project Setup:** The first step is to set up your Supabase project online. This involves defining your database tables (users, requests, etc.) and configuring your RLS policies within the Supabase dashboard.
- **Frontend Deployment:** Your React Native app will be deployed through **Expo EAS Build** (for a production-like build) or by running it locally during development.
- **Basic Testing:** Conduct thorough **manual testing** of the core user flows for each role:
    - **Volunteer:** Sign-up, view assigned task, check-in/out, send a chat message.
    - **Pilgrim:** Sign-up, create a request (e.g., lost child with photo), view assigned volunteer on map.
    - **Admin:** View dashboard counts, view new requests, manually assign a task, view volunteer status.
- **Debugging:** Utilize **Supabase logs** (available in the Supabase dashboard) to inspect server-side errors or function execution. For client-side issues, use your React Native app's development server logs and browser console (if web debugging).

By following this descriptive, MVP-focused guideline, you'll be able to build a functional and impressive prototype of BandhuConnect+ within your 3-day hackathon, clearly showcasing its core value proposition. Good luck!