BandhuConnect+ Frontend Guidelines: React Native Hackathon MVP (Descriptive) 🚀

This guideline provides a focused and detailed approach to building the frontend for your BandhuConnect+ MVP within a 3-day hackathon timeframe using React Native. It prioritizes core functionality, rapid UI development, and essential user experience elements for all three user types: Volunteers, Admins, and Pilgrims/Attendees.

1. Architectural Principles (MVP Focus)

For the hackathon, our frontend architecture will prioritize rapid development and core functionality over full-scale optimization or every edge case.

- **Cross-Platform Development:** Leverage React Native to build a single codebase that runs on both iOS and Android. This significantly speeds up development by avoiding platform-specific code, making it ideal for a hackathon.
- **Component-Based Architecture:** Utilize React Native's component-based nature to build reusable UI elements (e.g., buttons, input fields, task cards, map views). This promotes modularity and faster assembly of screens.
- **Intuitive User Experience (UX):** Focus on clear, simple, and intuitive interfaces for each user role. A hackathon prototype needs to be immediately understandable. Minimize complex navigation or excessive settings.
- **Core Functionality First:** Prioritize implementing the absolute essential features for each user type to demonstrate the application's value proposition. Visual polish and advanced features can be added post-hackathon.

2. Tech Stack & Tools (Leveraging React Native Ecosystem for MVP)

We'll choose tools that enable rapid development and address key requirements for a mobile MVP.

2.1. React Native with Expo

- **React Native:** The primary framework for building native mobile apps using JavaScript/TypeScript and React.
- **Expo:** An open-source platform for making universal native apps with React. It dramatically simplifies the development workflow (setup, local testing with Expo Go, build process) making it **essential for a hackathon**.
    - **Usage:** Use Expo CLI for project initialization and running the app.

2.2. UI Libraries & Styling

- **Rapid UI Development:** Avoid complex custom CSS setups. Use a utility-first CSS framework or a component library for speed.
    - use a styling approach like **Tailwind CSS for React Native** (e.g., with NativeWind or similar tools). This allows for quick, responsive styling directly in JSX.
- **Consistent Styling:** Define a basic color palette and typography once, and reuse it across the app for visual consistency.

2.3. Mapping Library

- **react-native-maps:** This library is the go-to for integrating maps into React Native apps.
    - **Usage:** We'll use it to display a map, add markers for locations (volunteer's current, task/request), and potentially animate a marker for simulated movement.
    - **Configuration:** Be aware that it requires API keys (e.g., Google Maps API) and native module setup, which Expo can simplify but still requires attention.

2.4. State Management

- **React Context API / useState:** For an MVP, React's built-in Context API combined with useState and useReducer hooks is generally sufficient for managing application state. Avoid external state management libraries like Redux unless absolutely necessary and your team is highly proficient, as they add setup overhead.
- **Supabase Client Integration:** The Supabase client library (@supabase/supabase-js) will be the primary tool for interacting with the backend (authentication, database queries, real-time subscriptions). This client library often works well with React Hooks.

3. Core UI/UX Principles (Hackathon Focus)

Design decisions should prioritize clarity and directness.

- **Mobile-First Design:** Ensure the layout and interactivity are optimized for small touchscreens. Large, tappable areas and clear fonts are essential.
- **Clear Calls to Action (CTAs):** Buttons and interactive elements should have unambiguous labels (e.g., "Sign Up," "Check In," "Report Issue").
- **Minimalist Interface:** Avoid clutter. Each screen should have a clear purpose. Only display essential information.
- **Loading States:** Provide visual feedback (e.g., loading spinners) when data is being fetched or an action is being processed, especially for map loading or API calls.

4. Feature Implementation Guidelines (Per User Role - MVP)

Here's how to approach the key features for each part of the app.

4.1. Volunteer App Features

- **Sign-Up Page:**
    - **UI:** Simple form with text inputs for Name, Email, Mobile No, Age.
    - **Skills:** Use checkboxes or a multi-select dropdown for Med, Tech, Logistics, Cleaning, General.
    - **Data Privacy & Consent:** A single checkbox "I agree to the terms" with basic text.
    - **Backend Interaction:** Use Supabase Auth to register the user and insert profile details (including skills, age) into the users table.
- **Dashboard:**
    - **Volunteer Display:** Prominently show the volunteer's name at the top.
    - **Task Display:** A card or section showing a hardcoded/randomized task. Include description. If it's a "Lost Person" task, display a placeholder image for the photo and hardcoded Name, Age, Phone Number.
    - **Duty Timings:** Display a hardcoded duty schedule.
    - **Map Display:** Integrate react-native-maps. Show two markers: one for the simulated volunteer's current location (can be a fixed point or a simple animated path for demo) and one for the hardcoded task location. The map should be the primary visual element in this section.
    - **Check-in-out:** Two distinct buttons ("Check In," "Check Out"). Clicking them should update the volunteer's status (e.g., in a status column in the users table or volunteer_tasks table) and visually reflect it (e.g., "On Duty," "Off Duty").
- **General Chatroom:**
    - **UI:** A basic message list and an input field with a send button.
    - **Backend Interaction:** Use Supabase Realtime to subscribe to the general chat channel and display new messages instantly. Use Supabase client to insert new messages into the messages table.

4.2. Admin Dashboard Features

- **Admin Dashboard:**
    - **UI:** Display placeholder counts for "Active Volunteers," "Inactive Volunteers," and "Distinct Services." These can be hardcoded or fetched with simple Supabase queries for a basic demo.
- **Request Panel:**
    - **UI:** A list of requests (e.g., cards for each request). Each card should show Request ID, User ID (pilgrim name if available), Type of Request, Service Activity.
    - **Map:** A separate screen or modal view that displays the request's location on a map (using react-native-maps) when a request is selected.
    - **Backend Interaction:** Use Supabase Realtime to listen for new requests. Use Supabase client to fetch request details and update their status.
- **Volunteer Panel:**
    - **UI:** A list of volunteers. Each item shows Vol ID, Status, current Task (if any), Skills.
    - **Manual Override (Simplified):** For the hackathon, implement simple buttons or toggles for "Change Status" and "Assign Task" (to a hardcoded task). The actual assignment logic can be basic and performed directly via Supabase client updates.

4.3. Pilgrim/Attendee App Features

- **Sign-Up Page:**
    - **UI:** Simple form for Name, Mobile No. Email can be optional.
    - **Data Privacy & Consent:** A mandatory checkbox for "I agree to the terms and privacy policy."
    - **Backend Interaction:** Use Supabase Auth to register the user and insert basic profile details into the users table.
- **Request Assistance:**
    - **UI:** A form with a dropdown/radio buttons for "Type of Request."
    - **Lost Child Field:** If "Lost Child" is selected, dynamically show fields for Photo Upload (using ImagePicker from Expo), Name, Age, Contact Phone Number.
    - **Location:** Display auto-detected GPS coordinates or a simple map view (using react-native-maps) to allow confirmation/manual adjustment.

    - **Description:** Text input for description. Voice recording is a stretch goal.
    - **Backend Interaction:** Insert the request into the requests table (including photo URL if uploaded).
- **Report Sanitation Issues:**
    - **UI:** Similar to Request Assistance, with auto-detected location, Photo Upload, and a Description field.
    - **Backend Interaction:** Insert the report into the requests table.
- **View Volunteer Status:**
    - **UI:** After submitting a request, display a screen showing "Volunteers on the Way" status.
    - **Map Tracking:** A dynamic map (using react-native-maps) showing the pilgrim's request location and a simulated moving marker for the assigned volunteer(s).
- **Language Selection:**
    - **UI:** A dropdown or settings screen to select preferred language.
    - **Implementation:** For MVP, this might just change a local state variable and display a message, or simply show the UI elements in a different hardcoded language to demonstrate the feature. Full internationalization (i18n) is a post-hackathon task.

5. API Interaction (Supabase Integration for MVP)

We'll primarily use the Supabase JavaScript client library.

- **Direct Supabase Client:** All frontend code will interact directly with Supabase via its client library (@supabase/supabase-js). This client handles authentication, database queries, and real-time subscriptions, making it very efficient.
- **Local State Management:** Fetch data from Supabase and store it in component-level state or React Context for display.
- **Error Handling:** Implement basic error handling using try...catch blocks around Supabase calls. Display user-friendly error messages (e.g., an alert or a small text message on the screen) for any failures.

6. Security & Data Handling (MVP Focus)

- **Client-Side Input Validation:** Implement basic validation for forms (e.g., required fields, email format) to improve user experience before sending data to the backend.
- **Secure Environment Variables:** Store your Supabase URL and anon key securely using **Expo's environment variable capabilities** (e.g., expo-constants or .env files with a build tool). Never hardcode these directly in your source code.
- **Image Handling:** For photo uploads, use Expo's ImagePicker to select images and Supabase client's storage upload functions to send them to your Supabase Storage bucket. Ensure RLS policies on Storage are set up.

7. Testing & Debugging (Hackathon Level)

- **Manual Testing:** Thoroughly test all core user flows on an emulator/simulator and, if possible, on a physical device using Expo Go.
- **Expo Go:** Utilize Expo Go for rapid iteration and debugging during development.
- **React Native Debugger:** Use tools like React Native Debugger (which combines React DevTools and Redux DevTools) for inspecting component state and network requests.
- **Supabase Logs:** Monitor logs in your Supabase dashboard for any backend errors or function execution issues.

By adhering to these guidelines, your team can build a functional, demonstrable, and impressive BandhuConnect+ MVP within the hackathon's 3-day window, effectively showcasing its core value proposition across all three user segments.