For a real-time, seamless mobile application designed for non-typing (tap-to-send) communication within a local church, meeting hall, or event space, there are several excellent tech stack options that can ensure near-instant messaging with robust offline/local network capabilities and modern user authentication. Here are tailored recommendations for front-end, back-end, and networking layers that match your requirements, especially prioritizing zero-lag and local network-based communication:

Front-End (Mobile UI/UX)

- **React Native or Flutter:** Both frameworks offer fast, cross-platform development for Android and iOS, support rapid UI prototyping for split-screen layouts, and integrate well with real-time messaging libraries.
    - **React Native:** Excellent community support, easier integration with web technologies, and good performance for lightweight real-time apps.
    - **Flutter:** Exceptional for beautiful UI, smooth animations, and fast rendering; suitable for event-driven interfaces.

Back-End / Real-Time Communication

- **Local Network Messaging (LAN support):**
    - **WebSocket-based server:** Enables real-time, bi-directional messaging ideal for event scenarios with minimal latency. Use frameworks like Node.js with `socket.io` for reliability and scalability.
    - **Peer-to-peer (P2P) LAN:** For ultra-local communication, frameworks/libraries like **PeerJS** or **Nearby Connections API** (Android) support direct device-to-device messaging over Wi-Fi, Wi-Fi Direct, or Bluetooth—great for on-site church and event scenarios where the internet isn't reliable.
- **Backend Framework:**
    - **Node.js with Express.js:** Lightweight, scalable, and integrates easily with WebSocket libraries.
    - **FastAPI (Python):** If you need REST endpoints or custom business logic.

Authentication and User Management

- **Firebase Authentication:** Simplifies user sign-in flows, including Google OAuth, and can store user types and profiles securely.
- **Optional:** Custom JWT (JSON Web Token) if you prefer privacy or local-only management.

Database

- **Firebase Realtime Database or Firestore:** Best for storing structured categories, subcategories, custom texts, and user profiles with real-time syncing.
- **SQLite:** For offline storage of app categories/messages on-device.

- **Redis/Memcached:** For caching and rapid message delivery if scaling up to multiple rooms/halls.

Network Layer & Real-Time Sync

- **WebSockets (socket.io, WebRTC):** For instant delivery—critical for your non-typing, category-based interactions.
- **Service Workers (PWA mode):** Optional, for push notifications or offline capabilities on web.

Deployment/Infrastructure

- **Docker:** Containerize your backend services for easy deployment and updates.
- **Nginx/Traefik:** Useful as reverse proxies to manage connections in local network mode.
- **Optional:** Raspberry Pi or mini server for LAN hosting (if deploying local-only, without cloud).

Optional Extras for Media, Band, and Admin Roles

- **Role-based UI:** Dynamic split-screen, showing categories/subcategories per user type with customization stored in DB.
- **Google OAuth Integration:** For auto sign-in, session persistence, and easy role switching.

Recommended Stack Table

Layer	Tech Option	Rationale
Front-End	React Native / Flutter	Fast dev, great UI, cross-platform, seamless real-time UX
Real-Time Sync	socket.io / WebSockets	Live, lag-free event messaging over LAN/Wi-Fi
P2P LAN	PeerJS / Nearby Connections	Ultra-local messaging, fallback if no router/internet
Backend	Node.js / Express.js	Lightweight, scalable, integrates with WebSockets
Auth	Firebase Auth / JWT	Easy login, role management, session recall
Database	Firestore / Firebase DB	Real-time data sync for categories/messages
Offline DB	SQLite	Stores categories/custom texts locally
Hosting	Docker (LAN server)	Fast deploy, easy config; LAN hosting option

---

Tech Stack Selection Guidance

- For single-hall, truly LAN-only setups, prioritize WebSocket/P2P options and lightweight local server hosting.

- For multiple halls/events, or cloud sync, use Firebase backend with socket.io plus Node.js for ultra-reliable real-time performance.
- For custom user types/roles, design modular UI using React Native or Flutter, pulling custom category data from Firestore.

This stack ensures **real-time, lag-free communication**, role-based UI, and reliable authentication, ideal for your non-screaming, tap-to-send event app needs.