he Nexus Network – Complete Plan

1. Core Idea

The Nexus Network is a **community-driven platform** where users, communities, and events interact in a structured, role-based environment. The platform supports:

- Thousands of users
- Multiple communities
- Role-based access and moderation
- Posts, events, and RSVPs
- Admin and staff controls

Think of it like a **mix between LinkedIn communities, Facebook groups, and event platforms**, optimized for community engagement and management.

2. User Roles & Permissions

Role	Permissions / Access
Admin	Full access. Manage users, communities, posts, events, system settings.
Staff	Moderate content, assist admin, basic app management.
Community Lead	Create and manage communities, posts, events, and volunteers under their community.
Volunteer	Manage specific tasks in communities, post updates, help organize events.
User	View feed, join communities, RSVP for events, comment on posts.

3. Key Features

A. Feed

- Displays posts from communities user is part of and global posts.
- Users can **like, comment, share**.
- Posts can contain **text, images, or links**.

- **Filtering** by community, popularity, or time.

B. Profile

- Personal profile with **bio, avatar, role, communities**.
- Users can **create posts** from their profile page.
- Option to **edit profile** and manage settings.

C. Communities

- Community pages for each group.
- Community lead and volunteers can **manage posts, members, and events**.
- Members can **join/leave, view posts, and participate in discussions**.

D. Events

- Communities can create events (meetups, workshops).
- Users can **RSVP** (going, interested, not going).
- Events appear on **community page** and optionally **feed**.

E. Admin & Staff Dashboard

- **User management:** Promote/demote, ban, or manage roles.
- **Community approval:** Approve or reject community creation.
- **Event oversight:** Manage or moderate all events.
- **Reports & stats:** Total users, active communities, posts, and events.

F. Notifications

- New posts in communities user is part of
- Event reminders / RSVP updates
- Role updates (e.g., promoted to volunteer/community lead)

G. Optional Future Features

- Private messaging between users
- Media uploads (images/videos) in posts
- Analytics dashboard for admins (engagement, growth trends)
- Integration with email/SMS for event notifications

4. App Flow

1. Entry / Authentication

- User visits /login → authenticates → redirected based on role:
  - Admin → /admin/dashboard
  - Staff → /staff/dashboard
  - Community Lead / Volunteer / User → /feed

2. Feed Interaction

- See posts → Like / Comment → Post updates from profile or community page

3. Profile Management

- Edit profile → Update avatar → Create personal posts → View own activity

4. Community Interaction

- Browse communities → Join/Leave → Participate in posts/events → Community Leads manage content

5. Events

- View upcoming events → RSVP → Attend / track participation → Volunteers manage community events

6. Admin / Staff Control

- Dashboard for monitoring all activities
- Approve communities, moderate posts, manage users

5. Database Structure (High-level)

1. **users**
   - id, name, email, password, role, avatar, created_at
2. **posts**
   - id, user_id, community_id (nullable), content, media_url, created_at
3. **communities**
   - id, name, description, lead_id, created_at
4. **community_members**
   - id, community_id, user_id, role
5. **events**
   - id, community_id, title, description, schedule, created_by
6. **event_rsvp**
   - id, event_id, user_id, status
7. **notifications**
   - id, user_id, type, message, read_status, created_at

6. Frontend Architecture

- **HTML, CSS, JS** for UI
- **Components**: Navbar, Footer, PostCard, CommunityCard, EventCard
- **Pages**: Login, Register, Feed, Profile, Community, Events, Admin, Staff
- **API Layer**: api.js → handles all fetch calls to backend
- **Role-based Guards**: authGuard.js → restrict access based on user role

7. Backend Architecture

- **Node.js + Express** (or any other backend later)
- **API Routes**:
  - /api/auth → login/register
  - /api/users → fetch/update users
  - /api/posts → CRUD posts
  - /api/communities → CRUD communities
  - /api/events → CRUD events
  - /api/rsvp → RSVP events
- **Middleware**:
  - authMiddleware → verify login token
  - roleMiddleware → enforce role-based permissions
- **Database**: PostgreSQL (can migrate to Supabase or managed cloud DB later)

8. Future-proofing & Migration

- Keep **frontend independent** of backend specifics → API calls only
- Database schemas compatible with Postgres → easy migration to Supabase or cloud DB
- Role-based middleware for **security and modularity**
- Frontend modular components → easy to redesign or add pages