Alumni Connect Hub - Complete Project Prompt Document

> **Version:** 1.0
> **Type:** Master Development Specification
> **Purpose:** Complete guide to build a multi-tenant university alumni networking platform

🎯 Project Overview

Build a **comprehensive, modern alumni networking platform** that connects university alumni globally. The platform should be a multi-tenant SaaS solution where each university has its own customized instance with unique branding, while sharing the same codebase.

Core Mission

Create a LinkedIn-style professional networking platform specifically for university alumni, with features for social networking, mentorship, events, career development, and community building.

📋 Executive Requirements

What to Build

A full-featured Progressive Web App (PWA) for university alumni that includes:

- Social feed with posts, comments, likes, and sharing
- Alumni directory and networking (connection requests)
- Events management (virtual & in-person)
- Groups and communities
- Real-time messaging system
- Mentorship matching (Tinder-style swipe interface)
- Document request system (transcripts, recommendations)
- AI-powered career roadmap generator

- AI resume and cover letter generator
- Admin panel for university management
- Super admin panel for platform management
- Multi-tenant architecture with custom branding per university
- Progressive Web App capabilities (installable, offline support)

Target Users

1. **Alumni** - Graduates who want to network, mentor, find opportunities
2. **University Admins** - Manage their university's alumni community
3. **Platform Super Admins** - Manage multiple universities and platform

🏗️ Technical Stack Requirements

Frontend Framework

- **React 18** with TypeScript (strict mode)
- **Vite** as build tool (for fast HMR)
- **React Router v6** for routing

UI & Styling

- **TailwindCSS** for utility-first styling
- **shadcn/ui** component library (50+ components)
- Support for dark/light themes
- Fully responsive (mobile-first design)

State Management

- **React Context API** for global state (no Redux)
- Multiple context providers:
  - AuthContext (user authentication & sessions)
  - ThemeContext (dark/light mode)
  - UniversityContext (multi-tenant branding)
  - EventsContext (events management)
  - GroupsContext (groups management)
  - ConnectionsContext (alumni networking)

    - SidebarContext (UI state)

Additional Libraries

- **React Hook Form** - Form management
- **Zod** - Form validation
- **Lucide React** - Icon library
- **Recharts** - Data visualization
- **TanStack Query** - Server state management
- **date-fns** - Date handling
- **Sonner** - Toast notifications

PWA Requirements

- Service worker for offline support
- Installable on desktop and mobile
- Direct login access when opened as installed app
- Custom install prompt
- App icons (192x192, 512x512)
- Manifest.json with proper configuration

🎨 Design Requirements

Visual Design

- Modern, clean, professional interface
- Gradient accents (purple/blue themes)
- Card-based layout
- Smooth animations and transitions
- Glass-morphism effects where appropriate
- Consistent spacing using Tailwind scale

Responsive Breakpoints

- Mobile: < 768px (bottom navigation)
- Tablet: 768px - 1024px (optional sidebar)
- Desktop: > 1024px (full sidebar navigation)

Navigation Structure

- **Mobile:** Bottom navigation bar (fixed)
- **Desktop:** Left sidebar with collapsible sections
- Both show same navigation items based on user role

Theme Support

- Light mode (default)
- Dark mode (toggle in header)
- Respect system preferences
- Persist user preference in localStorage

👥 User Roles & Permissions

1. Alumni (Regular Users)

Access to:

- ✅ View and create posts (text, images, jobs, announcements)
- ✅ Comment and like posts
- ✅ Browse events and register
- ✅ Join and create groups
- ✅ Connect with other alumni (send/accept requests)
- ✅ Send and receive messages
- ✅ Find mentors via swipe interface
- ✅ Request official documents (transcripts, letters)
- ✅ Generate AI resume and cover letters
- ✅ Get AI career roadmap guidance
- ✅ View alumni map (global heatmap)
- ✅ Search globally (users, posts, events, groups)
- ✅ Customize profile with rich information
- ✅ Receive notifications

Cannot Access:

- ❌ Admin panel
- ❌ User management

- ❌ Content moderation tools
- ❌ Platform analytics

2. Admin (University Level)

Access to:

- ✅ All alumni features
- ✅ User management (add, edit, remove, import CSV)
- ✅ Content moderation (approve, edit, delete posts)
- ✅ Event management (create, edit, approve)
- ✅ Group management (create, edit, moderate)
- ✅ Document request management (approve/reject)
- ✅ Password reset requests handling
- ✅ Mentor assignment and verification
- ✅ Fundraiser management
- ✅ Knowledge base management
- ✅ University branding customization (logo, colors)
- ✅ Analytics for their university
- ✅ Simplified profile page (not alumni profile)

Cannot Access:

- ❌ Super admin features
- ❌ Other universities' data
- ❌ Platform-wide settings
- ❌ Create/manage universities

3. Super Admin (Platform Level)

Access to:

- ✅ All admin features (all universities)
- ✅ Create and manage universities
- ✅ Manage university admins
- ✅ Platform-wide user management
- ✅ Platform-wide analytics
- ✅ Advertisement management
- ✅ Global password reset handling
- ✅ Platform configuration

- ✅ View cross-university metrics
- ✅ Simplified profile page

📱 Page-by-Page Specifications

PUBLIC PAGES

1. Landing Page (`/`)

Purpose: Marketing page for the platform

Sections:

- Hero section with gradient background
  - Headline: "Connect. Network. Grow."
  - Subheadline: "Join your university's alumni community"
  - CTA buttons: "Get Started" and "Learn More"
  - Animated background or graphic
- Features section (cards with icons)
  - Feature 1: "Network with Alumni"
  - Feature 2: "Attend Events"
  - Feature 3: "Find Mentors"
  - Feature 4: "Career Growth"
  - Feature 5: "Stay Connected"
  - Feature 6: "Give Back"
- CTA section
  - Strong call-to-action to join
  - Button to login or register
- Footer
  - Copyright
  - Links (Privacy, Terms, Contact)
  - Social media icons

Special Behavior:

- If opened as installed PWA, redirect immediately to `/login`
- Show PWA install prompt if not installed (dismissible)

2. Login Page (`/login`)

Purpose: Authentication gateway

Form Fields:

- Email (required, email validation)
- Password (required, min 6 chars)
- "Remember me" checkbox (optional)

Elements:

- Logo/branding at top
- "Forgot Password?" link
- "Login" button (primary)
- Error messages display area

Behaviors:

- Validate fields on blur and submit
- Show loading state on submit
- On success: redirect to:
  - Alumni → `/dashboard`
  - Admin → `/admin`
  - Super Admin → `/superadmin`
- On failure: show error message
- Redirect authenticated users away from this page

3. Forgot Password (`/forgot-password`)

Purpose: Password recovery

Elements:

- Email input field
- "Send Reset Link" button
- Back to login link

Behaviors:

- Send password reset request
- Show success message
- Admin/Super admin requests go to admin dashboard for handling

ALUMNI PAGES

4. Dashboard (`/dashboard`)

Purpose: Main social feed (like Facebook/LinkedIn feed)

Layout:

- Desktop: 3-column (left nav, center feed, right sidebar)
- Mobile: Single column feed

Components:

- Create post button/modal trigger (top)
- Filter tabs: All Posts, My Posts, Following
- Post feed (infinite scroll)
  - Each post shows:
    - Author avatar, name, graduation year
    - Post content (text, image, video)
    - Post type badge (Job, Event, Announcement)
    - Like button (with count)
    - Comment button (with count)
    - Share button
    - Edit/Delete (if own post)
    - Timestamp
  - Comment section (expandable)
    - Show top 2 comments
    - "View all X comments" button
    - Comment input (if logged in)
- Right sidebar:
  - Profile summary card
  - Suggested connections
  - Upcoming events
  - Active groups

Post Types:

- Text post
- Image post

- Job posting (with title, company, location)
- Event promotion
- Announcement (highlighted style)

5. Profile Page (`/profile`)

Purpose: View and edit user profile

Sections:

- Header
  - Cover photo (editable)
  - Profile picture (editable)
  - Name and headline
  - Edit profile button (own profile)
  - Connect button (other profiles)
  - Message button (if connected)
- About section
  - Bio/summary
  - Current position
  - Location
  - Website/social links
- Education
  - University
  - Degree
  - Major
  - Graduation year
  - GPA (optional)
- Work Experience
  - Multiple entries
  - Title, company, dates, description
  - Add/edit/remove buttons
- Skills
  - Skill tags
  - Add/remove skills
  - Endorsements (future)
- Portfolio/Projects
  - Project cards
  - Title, description, links

- Posts
  - User's recent posts
  - Filter by type

Edit Modal:

- Form with all editable fields
- Image upload for profile/cover
- Save and cancel buttons
- Real-time preview

6. Events Page (`/events`)

Purpose: Browse and manage events

Tabs:

- All Events (upcoming)
- Registered Events
- Past Events

Event Card Layout:

- Event banner image
- Title
- Date & time
- Location (virtual/physical)
- Type badge (Networking, Career, Social, Webinar)
- Registration button
- Attendee count
- Brief description

Event Details Modal:

- Full event information
- Organizer details
- Full description
- Registration form
- Add to calendar button
- Share button

Create Event Modal (Admin/Alumni):

- Title (required)
- Description (required)
- Date & time (date picker)
- Location (text)
- Event type (dropdown)
- Virtual meeting link (if virtual)
- Capacity (optional)
- Banner image upload
- Create button

7. Groups Page (`/groups`)

Purpose: Browse and manage group memberships

Tabs:

- All Groups
- My Groups
- Create Group

Group Card:

- Group image/icon
- Group name
- Member count
- Description preview
- Category badge
- Join/Leave button
- Privacy indicator (Public/Private)

Group Details Page:

- Group header
  - Cover photo
  - Name, description
  - Member count
  - Join/Leave button
  - Share button
- Discussion feed (like mini dashboard)
- Members list
- Events (group-specific)

- Files/Resources

Create Group Modal:

- Group name (required)
- Description (required)
- Category (dropdown: Academic, Professional, Social, etc.)
- Privacy setting (Public/Private)
- Group image upload
- Create button

8. Connections Page (`/connections`)

Purpose: Manage alumni network

Tabs:

- My Connections (approved)
- Received Requests (pending)
- Sent Requests (pending)
- Suggestions (recommended)

Connection Card:

- Profile picture
- Name
- Headline
- Graduation year
- Mutual connections count
- Action buttons:
  - Message (if connected)
  - Connect / Accept / Cancel
  - Remove connection

Features:

- Search/filter connections
- Sort by: Name, Year, Location
- Bulk actions (select multiple)

9. Chat/Messaging (`/chat`)

Purpose: Real-time messaging

Layout:

- Left: Conversation list
  - Search conversations
  - List of chats (sorted by recent)
  - Each shows: Avatar, name, last message, timestamp
  - Unread indicator
- Right: Active conversation
  - Chat header (name, online status)
  - Message history (scrollable)
  - Message input with send button
  - Typing indicators
  - Emoji picker (optional)

Message Types:

- Text messages
- Image sharing
- File sharing (future)
- Link previews

Features:

- Real-time updates (polling or WebSocket)
- Read receipts
- Search messages
- Group chats

10. Mentorship Match (`/mentorship`)

Purpose: Tinder-style mentor matching

Layout:

- Card stack interface
  - Large mentor card in center
  - Profile photo
  - Name, current role

    - Expertise tags
    - Bio/experience summary
    - Match score (AI-powered, e.g., "95% match")
    - Availability indicator

Actions:

- ❌ Pass (swipe left / X button)
- ✅ Match (swipe right / ✓ button)
- ℹ️ View Full Profile (info button)

After Match:

- Celebration animation
- "It's a match!" screen
- Send message button
- View matches tab

Tabs:

- Discover (swipe interface)
- My Matches (matched mentors)
- Requests (if you're a mentor)

11. Documents Page (`/documents`)

Purpose: Request and generate documents

Tabs:

Tab 1: Request Official Documents

- Form to request:
    - Transcript
    - Recommendation letter
    - Verification letter
    - Enrollment certificate
- Request details:
    - Document type (dropdown)
    - Purpose (text)
    - Delivery method (Email/Mail)
    - Additional notes

- Submit button
- Request history table
  - Date, type, status, download link

Tab 2: AI Document Generator

- Sub-tabs:
  - Generate Resume
  - Generate Cover Letter

Resume Generator:

- Input fields:
  - Target role
  - Industry
  - Experience level
  - Key skills (comma-separated)
- "Generate Resume" button
- Preview area
- Download as PDF button

Cover Letter Generator:

- Input fields:
  - Company name
  - Position applying for
  - Key qualifications
  - Why interested
- "Generate Cover Letter" button
- Preview area
- Download as PDF button

Tab 3: History

- All generated and requested documents
- Download buttons
- Regenerate options

12. AI Career Roadmap (`/roadmap`)

Purpose: Personalized career guidance

Interface:

- Career goal input
  - "What career path are you interested in?"
  - Text area for goals
- Generate button
- Roadmap visualization
  - Timeline view
  - Milestones (with descriptions)
  - Skills to acquire
  - Recommended courses
  - Estimated timeframes
  - Industry insights
- Save/export roadmap button
- Share button

Roadmap Sections:

- Current Position
- Short-term goals (6 months)
- Mid-term goals (1-2 years)
- Long-term goals (3-5 years)
- Skills gap analysis
- Learning resources
- Networking recommendations

13. Notifications Page (`/notifications`)

Purpose: View all notifications

Notification Types:

- Connection requests
- Post interactions (likes, comments)
- Event reminders
- Group invitations
- Message notifications
- Admin announcements

Notification Card:

- Icon (based on type)
- Message text
- Timestamp
- Action button (View, Accept, etc.)
- Mark as read button

Features:

- Filter by type
- Mark all as read
- Delete notifications
- Real-time updates

ADMIN PAGES

14. Admin Dashboard (`/admin`)

Purpose: Overview of university administration

Widgets:

- Total users count (card)
- Active users today (card)
- Total events (card)
- Total groups (card)
- Pending document requests (card)
- Pending password resets (card)
- Recent activity feed
- Quick actions:
  - Add user
  - Create event
  - Moderate content
  - Customize branding

Charts:

- User growth over time (line chart)
- User engagement (bar chart)
- Post activity (area chart)

15. Admin Users Page (`/admin/users`)

Purpose: Manage alumni users

Features:

- User table with columns:
  - Profile pic
  - Name
  - Email
  - Graduation year
  - Status (Active/Inactive)
  - Last login
  - Actions (Edit, Delete, View)
- Search and filter
- Bulk import (CSV)
- Add single user form
- Edit user modal
- Delete confirmation

Bulk Import:

- CSV file upload
- Column mapping interface
- Preview before import
- Import progress indicator

16. Admin Events Page (`/admin/events`)

Purpose: Manage all events

Features:

- All events table
- Pending approval queue
- Create event button
- Edit/Delete events
- Approve/Reject events
- Event analytics

17. Admin Groups Page (`/admin/groups`)

Purpose: Manage groups

Features:

- Groups table
- Create group
- Edit/Delete groups
- Moderate group content
- View group analytics

18. Admin Feed Management (`/admin/feed`)

Purpose: Moderate content

Features:

- All posts table
- Filter by: Type, Status, Date
- Approve/Reject posts
- Edit posts
- Delete posts
- Pin important posts
- Feature posts

19. Admin Mentors (`/admin/mentors`)

Purpose: Manage mentor program

Features:

- Mentor list
- Verify mentors
- Assign mentor badges
- Mentor/mentee matching data
- Mentor applications approval

20. Admin Documents (`/admin/documents`)

Purpose: Handle document requests

Features:

- Pending requests table
- Approve/Reject buttons
- Upload processed document
- Request history
- Download templates

21. Admin Password Resets (`/admin/passwords`)

Purpose: Handle password reset requests

Features:

- Reset requests table
- Verify user identity
- Send reset links
- Manual password reset

22. Admin Branding (`/admin/branding`)

Purpose: Customize university appearance

Features:

- Logo upload (display on all pages)
- Primary color picker (affects buttons, links)
- Secondary color picker (affects accents)
- Accent color picker
- University name
- Preview of changes
- Reset to defaults button
- Save changes button

Preview Section:

- Shows sample page with new branding
- Real-time color updates

23. Admin Fundraiser (`/admin/fundraiser`)

Purpose: Manage fundraising campaigns

Features:

- Active campaigns list
- Create campaign
- Set goals and deadlines
- Track donations
- Campaign analytics

24. Admin Knowledge Base (`/admin/knowledge`)

Purpose: Manage FAQ and resources

Features:

- Articles list
- Create/edit articles
- Categories management
- Search functionality

SUPER ADMIN PAGES

25. Super Admin Dashboard (`/superadmin`)

Purpose: Platform-wide overview

Metrics:

- Total universities
- Total users (all universities)
- Total admins
- Platform revenue (future)
- Active users today
- System health indicators

Charts:

- Platform growth
- University comparison
- User distribution by university

26. Super Admin Universities (`/superadmin/universities`)

Purpose: Manage universities

Features:

- Universities table
  - Name
  - Admin count
  - User count
  - Status (Active/Inactive)
  - Created date
  - Actions
- Add university form
  - Name
  - Domain
  - Admin email
  - Logo
  - Primary color
- Edit university
- Activate/Deactivate
- Delete university

27. Super Admin Users (`/superadmin/users`)

Purpose: Manage all users across platform

Features:

- All users table (filterable by university)
- Global search
- View user details
- Suspend/Activate users
- User analytics

28. Super Admin Admins (`/superadmin/admins`)

Purpose: Manage university admins

Features:

- All admins table

- Assign admin role
- Remove admin role
- View admin activity
- Admin permissions

29. Super Admin Ads (`/superadmin/ads`)

Purpose: Manage platform advertisements

Features:

- Ad campaigns table
- Create ad
  - Title
  - Image
  - Link
  - Target universities
  - Start/end dates
- Edit/Delete ads
- Ad performance metrics

30. Super Admin Analytics (`/superadmin/analytics`)

Purpose: Platform analytics

Features:

- User engagement metrics
- University performance
- Feature usage statistics
- Revenue analytics (future)
- Export reports

31. Super Admin Password Resets (`/superadmin/passwords`)

Purpose: Handle platform-wide password issues

Features:

- Similar to admin passwords but for all universities
- Escalated cases

🧩 Component Specifications

Navigation Components

Navigation (`Navigation.tsx`)

- Wrapper that shows DesktopNav or MobileNav based on screen size
- Uses `isMobile` hook

DesktopNav (`DesktopNav.tsx`)

Layout:

- Fixed left sidebar
- Full height
- Collapsible (optional)
- Logo at top
- Navigation links with icons
- User profile section at bottom
- Theme toggle
- Logout button

Navigation Items (Alumni):

- Dashboard (Home icon)
- Profile (User icon)
- Events (Calendar icon)
- Groups (Users icon)
- Connections (Network icon)
- Chat (Message icon)
- Mentorship (Heart icon)
- Documents (FileText icon)
- Roadmap (Map icon)
- Notifications (Bell icon with badge)

Navigation Items (Admin):

- Dashboard
- Users
- Events

- Groups
- Feed
- Documents
- Mentors
- Passwords
- Branding
- Fundraiser
- Knowledge Base

Navigation Items (Super Admin):

- Dashboard
- Universities
- Users
- Admins
- Ads
- Analytics
- Passwords

MobileNav (`MobileNav.tsx`)

Layout:

- Fixed bottom navigation bar
- 5 main items (most important)
- Icons only (no text)
- Active indicator
- More menu for additional items

Main Items:

- Home/Dashboard
- Events
- Groups
- Chat
- Profile/Menu

Feature Components

PostModal (`PostModal.tsx`)

Purpose: Create and edit posts

Form Fields:

- Post type selector (dropdown)
  - Regular post
  - Job posting
  - Event promotion
  - Announcement
- Content (textarea with rich text)
- Image upload (optional)
- Video URL (optional)
- Job-specific fields (if job post):
  - Job title
  - Company
  - Location
  - Job type
- Visibility (Public/Connections only)

Actions:

- Cancel
- Save draft
- Post

EventModal (`EventModal.tsx`)

Purpose: Create events

Form Fields:

- Event title (required)
- Description (rich text)
- Date and time (date picker)
- End date/time
- Location (text or virtual)
- Virtual link (if virtual event)

- Event type (dropdown)
- Capacity (number)
- Registration deadline
- Banner image upload

GroupModal (`GroupModal.tsx`)

Purpose: Create groups

Form Fields:

- Group name
- Description
- Category
- Privacy setting
- Group image upload
- Tags

CommentSection (`CommentSection.tsx`)

Purpose: Display and add comments

Features:

- List of comments
  - Avatar
  - Name
  - Comment text
  - Timestamp
  - Like button
  - Reply button (optional)
- Comment input
  - Text area
  - Submit button
- Load more comments
- Comment count

NotificationBell (`NotificationBell.tsx`)

Purpose: Notification indicator

Features:

- Bell icon in header
- Unread count badge
- Dropdown with recent notifications
- "View all" link
- Mark as read
- Real-time updates

GlobalSearchDropdown (`GlobalSearchDropdown.tsx`)

Purpose: Universal search

Features:

- Search input in header
- Dropdown with results as you type
- Categorized results:
  - Users
  - Posts
  - Events
  - Groups
- Click to navigate
- Keyboard navigation

UniversityChatbot (`UniversityChatbot.tsx`)

Purpose: AI assistant (optional)

Features:

- Floating button (bottom right)
- Chat interface
- Answer common questions
- Help with navigation

WorldMapHeatmap (`WorldMapHeatmap.tsx`)

Purpose: Alumni location visualization

Features:

- Interactive world map
- Heat map showing alumni density

- Click on regions to see alumni list
- Zoom and pan
- Legend

PWAInstallPrompt (`PWAInstallPrompt.tsx`)

Purpose: Encourage PWA installation

Features:

- Dismissible card (bottom right)
- "Install App" button
- "Not Now" button
- Only show on web (not in installed PWA)
- Remember dismissal

Admin Components

All admin components should have:

- Data tables with search, sort, filter
- CRUD operations (Create, Read, Update, Delete)
- Confirmation modals for destructive actions
- Success/error toast notifications
- Loading states
- Empty states

🔄 State Management (Context Specifications)

AuthContext

State:

```
{
  user: User | null;
  isAuthenticated: boolean;
  isAdmin: boolean;
  isSuperAdmin: boolean;
}
```

Methods:

- `login(email, password)` → authenticate user
- `logout()` → clear session
- `updateProfile(data)` → update user info
- `checkAuth()` → verify token validity

Persistence:

- Store user data in localStorage
- Store JWT token in localStorage
- Auto-logout on token expiration

ThemeContext

State:

```
{
  theme: "light" | "dark";
}
```

Methods:

- `toggleTheme()` → switch between light/dark
- `setTheme(theme)` → set specific theme

Persistence:

- Store in localStorage
- Apply class to document element

UniversityContext

State:

```
{
  universities: University[];
  currentUniversity: University | null;
}
```

Methods:

- `getUniversity(id)` → fetch university data
- `updateBranding(id, branding)` → update university appearance
- `getBranding(id)` → get custom colors/logo

Features:

- Load university branding based on user
- Apply custom CSS variables
- Switch between universities (super admin)

EventsContext

State:

```
{
  events: Event[];
  registeredEvents: number[];
}
```

Methods:

- `createEvent(data)` → create new event
- `registerForEvent(eventId)` → register user
- `unregisterFromEvent(eventId)` → unregister
- `getUpcomingEvents()` → fetch future events
- `getMyEvents()` → fetch user's registered events

GroupsContext

State:

```
{
  groups: Group[];
  joinedGroups: number[];
}
```

Methods:

- `createGroup(data)` → create new group
- `joinGroup(groupId)` → join group
- `leaveGroup(groupId)` → leave group
- `getGroups()` → fetch all groups
- `getMyGroups()` → fetch joined groups

ConnectionsContext

State:

```
{
  connections: User[];
  receivedRequests: User[];
  sentRequests: User[];
}
```

Methods:

- `sendRequest(userId)` → send connection request
- `acceptRequest(userId)` → accept request
- `rejectRequest(userId)` → reject request
- `removeConnection(userId)` → remove connection
- `getConnections()` → fetch connections
- `getSuggestions()` → get recommended connections

SidebarContext

State:

```
{
  isOpen: boolean;
}
```

Methods:

- `toggleSidebar()` → open/close
- `closeSidebar()` → force close
- `openSidebar()` → force open

🎯 Multi-Tenancy Architecture

How It Works

1. Each university has a unique ID and custom branding
2. Users belong to a specific university
3. Data is filtered by university (except super admin)
4. Custom branding (logo, colors) applied dynamically

Branding System

University Branding Object:

```
{
  id: number;
  name: string;
  logo: string; // URL to logo
  primaryColor: string; // Hex color
  secondaryColor: string;
  accentColor: string;
}
```

CSS Variables:

- Apply colors as CSS custom properties
- Update `--primary`, `--secondary`, `--accent`
- Logo replaces default in navigation

Implementation:

- UniversityContext loads user's university on login
- Apply branding in root component

- Admin can customize from `/admin/branding`
- Changes reflect immediately

📱 Progressive Web App (PWA) Implementation

Manifest Configuration

File: `public/manifest.json`

```
{
  "name": "AlumniHub - University Alumni Network",
  "short_name": "AlumniHub",
  "start_url": "/login",
  "display": "standalone",
  "background_color": "#ffffff",
  "theme_color": "#3B82F6",
  "icons": [
    {
      "src": "/icon-192.png",
      "sizes": "192x192",
      "type": "image/png"
    },
    {
      "src": "/icon-512.png",
      "sizes": "512x512",
      "type": "image/png"
    }
  ],
  "shortcuts": [
    {
      "name": "Dashboard",
      "url": "/dashboard",
      "icon": "/icon-192.png"
    }
  ]
}
```

Service Worker

File: `public/sw.js`

Requirements:

- Cache essential resources (JS, CSS, images)
- Network-first strategy for API calls
- Cache-first for static assets
- Offline fallback page
- Auto-update when new version available

PWA Behaviors

1. **Landing Page Detection:**
   - Detect if opened as PWA vs web browser
   - If PWA: redirect `/` → `/login` immediately
   - If web: show landing page normally
2. **Install Prompt:**
   - Show custom install prompt on web
   - Dismissible for session
   - Show install icon in browser
3. **Offline Support:**
   - Cache pages after first visit
   - Show offline indicator when no connection
   - Queue actions (like posting) when offline

App Icons

- 192x192px (standard)
- 512x512px (high-res)
- Gradient design matching brand
- Transparent background

🔐 Security Requirements

Authentication

- JWT tokens for session management
- Tokens stored in localStorage
- Auto-refresh before expiration
- Secure password hashing (backend)

Authorization

- Route guards check user role
- UI elements hidden based on permissions
- API calls validate user permissions (backend)

Data Protection

- Never expose sensitive data in URLs
- Sanitize user inputs
- Prevent XSS attacks
- HTTPS only in production

Role-Based Access Control (RBAC)

- Check user role before rendering pages
- Redirect unauthorized users
- Hide admin features from regular users
- Super admin can access everything

🚀 Performance Optimizations

Code Splitting

- Lazy load pages with React.lazy()
- Wrap with Suspense and loading spinner
- Split vendor bundles (React, UI components)

Image Optimization

- Lazy load images (loading="lazy")
- Responsive images with srcset
- WebP format when possible
- Image error handling with fallbacks

Caching

- Cache API responses (TanStack Query)
- Browser caching for static assets
- Service worker caching

Bundle Size

- Tree shaking enabled
- Remove unused CSS
- Compress with gzip
- Analyze bundle with vite-bundle-visualizer

🎨 UI/UX Guidelines

Consistency

- Use shadcn/ui components everywhere
- Consistent spacing (Tailwind scale)
- Consistent color usage
- Consistent typography

Feedback

- Loading states for async operations
- Success/error toasts for actions
- Disabled states for buttons
- Progress indicators

Empty States

- Show helpful messages when no data
- Provide call-to-action
- Use illustrations or icons

Error Handling

- User-friendly error messages
- Fallback UI for errors
- Retry mechanisms
- Error boundaries for React errors

Accessibility

- Semantic HTML
- ARIA labels where needed
- Keyboard navigation support
- Color contrast compliance
- Focus indicators

📊 Data Flow Architecture

Typical Flow (Example: Creating a Post)

1. User clicks "Create Post" button
2. PostModal opens
3. User fills form and submits
4. Component validates data
5. API call to backend POST /api/posts
6. Backend creates post, returns data
7. Context updates (or re-fetch)
8. Dashboard re-renders with new post
9. Success toast appears
10. Modal closes

State Updates

- Use Context for global state
- Local state (useState) for component-specific data
- Lift state up when needed
- Avoid prop drilling (use Context)

🧪 Testing Requirements

Unit Tests (Optional but Recommended)

- Test utility functions
- Test custom hooks
- Test context logic

Integration Tests

- Test user flows (login → dashboard → create post)
- Test API integration
- Test context interactions

Manual Testing Checklist

- Test all pages as different user roles
- Test responsive design (mobile, tablet, desktop)
- Test dark/light themes
- Test PWA installation
- Test offline functionality
- Test cross-browser compatibility

📦 Build & Deployment

Build Configuration

Vite Config:

```
export default defineConfig({
  plugins: [react()],
  build: {
    outDir: "dist",
    sourcemap: true,
  },
  server: {
    port: 3000,
  },
});
```

Environment Variables

```
VITE_API_URL=your_backend_url
VITE_APP_NAME=AlumniHub
```

Deployment Checklist

- [ ] Build production bundle: `npm run build`
- [ ] Test production build: `npm run preview`
- [ ] Ensure HTTPS enabled
- [ ] Configure service worker
- [ ] Test PWA installation
- [ ] Verify manifest.json paths
- [ ] Test on multiple devices
- [ ] Check performance (Lighthouse)
- [ ] Set up error tracking
- [ ] Configure analytics

Hosting Recommendations

- **Vercel** (recommended for Vite)
- **Netlify**
- **AWS Amplify**
- **Firebase Hosting**

📋 Development Checklist

Phase 1: Setup & Foundation ✅

- [ ] Initialize Vite + React + TypeScript project
- [ ] Install dependencies (TailwindCSS, shadcn/ui, etc.)
- [ ] Set up project structure (pages, components, contexts)
- [ ] Configure Tailwind and shadcn/ui
- [ ] Set up React Router
- [ ] Create basic layout components

Phase 2: Authentication & Core State ✅

- [ ] Create AuthContext
- [ ] Create ThemeContext
- [ ] Create UniversityContext
- [ ] Build login page
- [ ] Build landing page
- [ ] Implement route protection
- [ ] Add role-based redirects

Phase 3: Alumni Features ✅

- [ ] Build Dashboard (social feed)
- [ ] Create PostModal component
- [ ] Build Profile page
- [ ] Create Events page + EventModal
- [ ] Create Groups page + GroupModal
- [ ] Build Connections page
- [ ] Build Chat/Messaging page
- [ ] Create Mentorship matching page
- [ ] Build Documents page
- [ ] Create AI Roadmap page
- [ ] Build Notifications page

Phase 4: Navigation & Layout ✅

- [ ] Create DesktopNav component

- [ ] Create MobileNav component
- [ ] Create Navigation wrapper
- [ ] Add NotificationBell
- [ ] Add GlobalSearch
- [ ] Create Footer
- [ ] Implement responsive design

Phase 5: Admin Features ✅

- [ ] Build Admin Dashboard
- [ ] Create Admin Users page
- [ ] Create Admin Events page
- [ ] Create Admin Groups page
- [ ] Create Admin Feed management
- [ ] Create Admin Documents page
- [ ] Create Admin Mentors page
- [ ] Create Admin Password resets page
- [ ] Create Admin Branding page
- [ ] Create Admin Fundraiser page
- [ ] Create Admin Knowledge base page

Phase 6: Super Admin Features ✅

- [ ] Build Super Admin Dashboard
- [ ] Create Universities management page
- [ ] Create Global Users page
- [ ] Create Admins management page
- [ ] Create Ads management page
- [ ] Create Analytics page
- [ ] Create Password resets page

Phase 7: PWA Implementation ✅

- [ ] Create manifest.json
- [ ] Create service worker (sw.js)
- [ ] Add PWA meta tags
- [ ] Create app icons
- [ ] Create PWAInstallPrompt component
- [ ] Implement PWA detection and redirect

- [ ] Test installation on multiple devices

Phase 8: Polish & Optimization ✅

- [ ] Implement dark/light theme
- [ ] Add loading states everywhere
- [ ] Add error states
- [ ] Add empty states
- [ ] Optimize images
- [ ] Implement code splitting
- [ ] Test responsive design
- [ ] Add toast notifications
- [ ] Add confirmation dialogs
- [ ] Polish animations

Phase 9: Testing & QA

- [ ] Test as Alumni user
- [ ] Test as Admin
- [ ] Test as Super Admin
- [ ] Test all CRUD operations
- [ ] Test on mobile devices
- [ ] Test PWA installation
- [ ] Test offline mode
- [ ] Cross-browser testing
- [ ] Performance testing

Phase 10: Documentation & Deployment

- [ ] Write README
- [ ] Create architecture documentation
- [ ] Build production bundle
- [ ] Deploy to hosting
- [ ] Configure domain
- [ ] Set up monitoring

🎯 Success Criteria

Functional Requirements

- ✅ All user roles work correctly
- ✅ All pages accessible and functional
- ✅ CRUD operations work for all features
- ✅ Real-time features work (notifications, chat)
- ✅ Search functionality works
- ✅ File uploads work
- ✅ Forms validate correctly
- ✅ Multi-tenancy works (branding per university)
- ✅ PWA installs and works offline

Non-Functional Requirements

- ✅ Fast load times (< 3s initial load)
- ✅ Responsive on all devices
- ✅ Accessible (WCAG 2.1 compliance)
- ✅ Secure (authentication, authorization)
- ✅ Scalable architecture
- ✅ Maintainable code
- ✅ Well-documented
- ✅ Cross-browser compatible

🎓 Key Features Summary

Social Networking

- Post creation (text, images, jobs, announcements)
- Like, comment, share posts
- Social feed with filters
- User profiles with rich information
- Connection requests and management
- Real-time messaging

Events & Groups

- Event creation and registration
- Event calendar
- Virtual and in-person events
- Group creation and management
- Group discussions
- Public and private groups

Career Development

- Mentorship matching (Tinder-style)
- AI career roadmap generator
- AI resume generator
- AI cover letter generator
- Job postings
- Alumni directory

Administration

- User management (bulk import)
- Content moderation
- Event management
- Group management
- Document request handling
- Password reset management
- Custom branding per university

Platform Management

- Multi-university management
- Admin assignment
- Platform analytics
- Advertisement management
- Global user management

Technical Features

- Progressive Web App (installable)

- Offline support
- Dark/light theme
- Multi-tenant architecture
- Role-based access control
- Responsive design
- Real-time updates

📝 Notes for AI/Developers

Development Approach

1. **Start with core:** Authentication and routing first
2. **Build incrementally:** One page at a time
3. **Test frequently:** Test each feature as you build
4. **Reuse components:** Don't repeat yourself
5. **Follow patterns:** Maintain consistency

Code Style

- Use TypeScript strict mode
- Follow React best practices
- Use functional components (no class components)
- Use hooks (useState, useEffect, useContext, custom hooks)
- Keep components small and focused
- Extract reusable logic to custom hooks
- Comment complex logic

Component Organization

```
src/
├── pages/           # Page components (routes)
├── components/      # Reusable components
│   ├── admin/      # Admin-specific components
│   ├── superadmin/ # Super admin components
│   └── ui/         # shadcn/ui components
├── contexts/        # React Context providers
├── hooks/           # Custom React hooks
├── lib/             # Utility functions
└── data/            # Static data / constants
```

Naming Conventions

- **Components:** PascalCase (e.g., `PostModal.tsx`)
- **Contexts:** PascalCase + Context (e.g., `AuthContext.tsx`)
- **Hooks:** camelCase + use prefix (e.g., `use-mobile.tsx`)
- **Utilities:** camelCase (e.g., `utils.ts`)
- **Pages:** PascalCase, descriptive (e.g., `Dashboard.tsx`)

Git Workflow

- Commit frequently with clear messages
- Use feature branches
- Test before merging
- Keep main branch stable

🚀 Getting Started (Implementation Order)

1. **Set up project:**

   ```
   npm create vite@latest alumni-connect-hub -- --template react-ts
   cd alumni-connect-hub
   npm install
   ```

2. **Install all dependencies** from package.json specifications above
3. **Set up TailwindCSS and shadcn/ui:**

```
npx shadcn-ui@latest init
```

4. **Create folder structure** as specified above
5. **Build contexts first** (they're the foundation)
   - AuthContext
   - ThemeContext
   - UniversityContext
   - Others...
6. **Build layout components:**
   - Navigation
   - DesktopNav
   - MobileNav
7. **Build pages in order:**
   - Landing → Login → Dashboard → Profile → etc.
8. **Test each feature** as you build it
9. **Implement PWA features** after core functionality works
10. **Polish and optimize** before deployment

🎉 Final Notes

This prompt provides a complete specification for building a professional, production-ready alumni networking platform. Follow the guidelines, maintain code quality, and test thoroughly. The result should be a modern, scalable SaaS application that serves multiple universities with custom branding.

Key Principles:

- User experience is paramount
- Code quality matters
- Security is non-negotiable
- Performance is critical
- Accessibility is required
- Documentation is essential

Good luck building AlumniHub! 🚀

Document Version: 1.0
Last Updated: December 8, 2024
Status: Complete Development Specification
Target: AI Assistants, Development Teams, Product Managers

📞 Support

If implementing this project and need clarification:

- Refer to FRONTEND_ARCHITECTURE.md for technical details
- Refer to PWA_IMPLEMENTATION.md for PWA specifics
- Check individual component files for implementation examples
- Review shadcn/ui documentation for UI components
- Review React Router documentation for routing patterns

Happy coding! May your builds be fast and your bugs be few! 🎯