# MACRO-HARD Teams - Implementation Documentation

## 1. Git Repository Organization and Development Workflow

### Repository Structure

SoftwareFrameworksA1/  
├── client/ # Angular frontend application  
│ ├── src/  
│ │ ├── app/  
│ │ │ ├── components/ # Angular components  
│ │ │ ├── services/ # Angular services  
│ │ │ ├── guards/ # Route guards  
│ │ │ └── models/ # TypeScript interfaces  
│ │ └── ...  
│ ├── package.json  
│ └── angular.json  
├── server/ # Node.js/Express backend  
│ ├── server.js # Main server file  
│ └── package.json  
├── README.md  
└── DOCUMENTATION.md # This file

### Development Workflow

* Development uses frequent updates without branching as I found this to be easier when I was the only collaborator for the project
* Committing after each milestone I had set out for myself
* Hot reload: Angular CLI for frontend, Node.js using nodemon for live reload in the backend.

## 2. Data Structures

### Client-Side Data Structures (TypeScript Interfaces)

#### User Interface

interface User {  
 id: string;  
 username: string;  
 password: string;  
 email?: string;  
 roles: Role[];  
 groups: string[];  
}  
  
type Role = 'super' | 'super\_admin' | 'groupAdmin' | 'group\_admin' | 'user';

#### Group Interface

interface Group {  
 id: string;  
 name: string;  
 creatorId: string;  
 ownerId: string;  
 adminIds: string[];  
 memberIds: string[];  
}

#### Channel Interface

interface Channel {  
 id: string;  
 name: string;  
 groupId: string;  
 bannedUserIds: string[];  
}

#### API User Interface (for admin operations)

interface ApiUser {  
 id: string;  
 username: string;  
 roles: string[];  
 groups: string[];  
 email?: string;  
}

### Server-Side Data Structures (JavaScript Arrays)

#### Users Array

const users = [  
 {   
 id: 'u\_1',   
 username: 'super',   
 password: '123',   
 email: '',   
 roles: ['super', 'super\_admin'],   
 groups: []   
 }  
];

#### Groups Array

const groups = [  
 {  
 id: 'g\_1',  
 name: 'Test Group',  
 creatorId: 'u\_1',  
 ownerId: 'u\_1',  
 adminIds: ['u\_1'],  
 memberIds: ['u\_1', 'u\_2']  
 }  
];

#### Channels Array

const channels = [  
 {  
 id: 'c\_1',  
 name: 'General',  
 groupId: 'g\_1',  
 bannedUserIds: []  
 }  
];

#### Supporting Arrays

const groupInterests = []; // Join requests  
const reports = []; // Admin reports  
const messages = []; // Chat messages

## 3. Angular Architecture

### Components

* **AppComponent**: Main application component with routing
* **LoginComponent**: User authentication interface
* **RegisterComponent**: User registration interface
* **DashboardComponent**: Main user dashboard with groups and actions
* **GroupDetailComponent**: Detailed group view with channels and members
* **ChannelViewComponent**: Channel chat interface
* **AdminPanelComponent**: Super admin management interface
* **NavbarComponent**: Navigation bar component

### Services

* **AuthService**: Authentication and user management
* **ApiService**: HTTP communication with backend
* **StorageService**: Local storage management

### Models

* **User**: User data structure
* **Group**: Group data structure
* **Channel**: Channel data structure
* **Role**: User role types

### Routes

const routes: Routes = [  
 { path: '', redirectTo: '/dashboard', pathMatch: 'full' },  
 { path: 'login', component: LoginComponent },  
 { path: 'register', component: RegisterComponent },  
 {   
 path: 'dashboard',   
 component: DashboardComponent,   
 canActivate: [AuthGuard]   
 },  
 {   
 path: 'group/:gid',   
 component: GroupDetailComponent,   
 canActivate: [AuthGuard]   
 },  
 {   
 path: 'channel/:cid',   
 component: ChannelViewComponent,   
 canActivate: [AuthGuard]   
 },  
 {   
 path: 'admin',   
 component: AdminPanelComponent,   
 canActivate: [AuthGuard, RoleGuard]   
 }  
];

### Guards

* **AuthGuard**: Ensures user is authenticated
* **RoleGuard**: Ensures user has required role for admin access

## 4. Node Server Architecture

### Main Server File (server.js)

* **Express Application**: Main server instance
* **Middleware**: CORS, JSON parsing, static file serving
* **In-Memory Storage**: Arrays for data persistence during development
* **Route Organization**: Grouped by functionality

### Global Variables

const users = []; // User data  
const groups = []; // Group data  
const channels = []; // Channel data  
const groupInterests = []; // Join requests  
const reports = []; // Admin reports  
const messages = []; // Chat messages

### Module Structure

* **Express**: Web framework
* **CORS**: Cross-origin resource sharing
* **Built-in modules**: HTTP, path, etc.

### Functions

* **Authentication**: Login/register validation
* **Authorization**: Role-based access control
* **Data Management**: CRUD operations for all entities
* **Business Logic**: Group management, user promotion, etc.

## 5. Server-Side Routes

### Authentication Routes

POST /api/login  
- Parameters: { username, password }  
- Returns: { ok: boolean, user?: User, msg?: string }  
- Purpose: User authentication  
  
POST /api/register  
- Parameters: { username, password, email }  
- Returns: { ok: boolean, id?: string, msg?: string }  
- Purpose: User registration

### User Management Routes

GET /api/users  
- Parameters: None  
- Returns: User[] (safe, no passwords)  
- Purpose: Get all users  
  
GET /admin/users  
- Parameters: { adminId }  
- Returns: User[] (super admin only)  
- Purpose: Admin user management  
  
DELETE /api/users/:userId  
- Parameters: { adminId }  
- Returns: { ok: boolean, msg: string }  
- Purpose: Delete user (super admin only)  
  
POST /api/users/:userId/promote-super  
- Parameters: { adminId }  
- Returns: { ok: boolean, msg: string }  
- Purpose: Promote to super admin  
  
DELETE /api/users/:userId/self  
- Parameters: { password }  
- Returns: { ok: boolean, msg: string }  
- Purpose: User self-deletion

### Group Management Routes

POST /api/groups  
- Parameters: { name, creatorId }  
- Returns: { ok: boolean, group?: Group, msg?: string }  
- Purpose: Create new group  
  
GET /api/users/:userId/groups  
- Parameters: None  
- Returns: Group[]  
- Purpose: Get user's groups  
  
POST /api/groups/:groupId/members  
- Parameters: { userId, adminId }  
- Returns: { ok: boolean, msg: string }  
- Purpose: Add user to group  
  
DELETE /api/groups/:groupId/members/:userId  
- Parameters: { adminId }  
- Returns: { ok: boolean, msg: string }  
- Purpose: Remove user from group  
  
DELETE /api/groups/:groupId  
- Parameters: { adminId }  
- Returns: { ok: boolean, msg: string }  
- Purpose: Delete group  
  
POST /api/groups/:groupId/interest  
- Parameters: { userId }  
- Returns: { ok: boolean, msg: string }  
- Purpose: Request to join group  
  
GET /api/groups/:groupId/interests  
- Parameters: { adminId }  
- Returns: Interest[]  
- Purpose: Get pending join requests  
  
POST /api/groups/:groupId/interests/:interestId/approve  
- Parameters: { adminId }  
- Returns: { ok: boolean, msg: string }  
- Purpose: Approve join request  
  
DELETE /api/groups/:groupId/interests/:interestId  
- Parameters: { adminId }  
- Returns: { ok: boolean, msg: string }  
- Purpose: Reject join request

### Channel Management Routes

POST /api/groups/:groupId/channels  
- Parameters: { name, adminId }  
- Returns: { ok: boolean, channel?: Channel, msg?: string }  
- Purpose: Create channel  
  
DELETE /api/groups/:groupId/channels/:channelId  
- Parameters: { adminId }  
- Returns: { ok: boolean, msg: string }  
- Purpose: Delete channel  
  
POST /api/channels/:channelId/ban  
- Parameters: { userId, adminId }  
- Returns: { ok: boolean, msg: string }  
- Purpose: Ban user from channel  
  
DELETE /api/channels/:channelId/ban/:userId  
- Parameters: { adminId }  
- Returns: { ok: boolean, msg: string }  
- Purpose: Unban user from channel

### Chat Routes

GET /api/channels/:channelId/messages  
- Parameters: { userId }  
- Returns: Message[]  
- Purpose: Get channel messages  
  
POST /api/channels/:channelId/messages  
- Parameters: { userId, content }  
- Returns: { ok: boolean, msg: string }  
- Purpose: Send message to channel

### Reporting Routes

POST /api/reports  
- Parameters: { reporterId, subject, message, type, relatedUserId }  
- Returns: { ok: boolean, msg: string }  
- Purpose: Create admin report  
  
GET /api/reports  
- Parameters: { adminId }  
- Returns: Report[]  
- Purpose: Get all reports (super admin only)

### Admin Routes

PATCH /admin/users/:id/role  
- Parameters: { add/remove, adminId }  
- Returns: { ok: boolean, msg: string }  
- Purpose: Toggle group admin role  
  
POST /admin/groups/:groupId/members  
- Parameters: { userId, adminId }  
- Returns: { ok: boolean, msg: string }  
- Purpose: Add user to group (super admin)  
  
DELETE /admin/groups/:groupId/members/:userId  
- Parameters: { adminId }  
- Returns: { ok: boolean, msg: string }  
- Purpose: Remove user from group (super admin)

## 6. Client-Server Interaction Details

### Authentication Flow

1. **Login**: Client sends credentials → Server validates → Returns user data → Client stores in session
2. **Registration**: Client sends user data → Server creates user → Returns success → Client redirects to login

### Dashboard Updates

1. **Load Groups**: Client requests user groups → Server filters by user ID → Returns groups → Client displays
2. **Create Group**: Client sends group data → Server creates group → Returns group → Client updates list
3. **Browse Groups**: Client requests all groups → Server returns filtered list → Client shows available groups

### Group Management

1. **Join Request**: Client sends interest → Server stores request → Returns success → Client shows confirmation
2. **Approve/Reject**: Admin clicks action → Client sends request → Server updates group → Returns success → Client refreshes
3. **Remove User**: Admin clicks remove → Client sends request → Server removes from group → Returns success → Client updates list

### Channel Operations

1. **Create Channel**: Admin fills form → Client sends data → Server creates channel → Returns channel → Client updates list
2. **Send Message**: User types message → Client sends to server → Server stores message → Returns success → Client updates chat
3. **Ban User**: Admin selects user → Client sends ban request → Server updates channel → Returns success → Client updates UI

### Admin Panel Operations

1. **Promote User**: Admin clicks promote → Client sends request → Server updates user roles → Returns success → Client refreshes list
2. **Delete User**: Admin confirms deletion → Client sends request → Server removes user → Returns success → Client updates list
3. **View Reports**: Admin loads panel → Client requests reports → Server returns reports → Client displays in sidebar

### Real-Time Updates

* **Group Changes**: Server updates group data → Client polls for updates → UI reflects changes
* **Message History**: Client requests messages → Server returns last 50 → Client displays in chat
* **User Status**: Server updates user roles → Client refreshes user data → UI shows new permissions

### Error Handling

* **Validation Errors**: Server returns error message → Client displays in UI
* **Permission Errors**: Server returns 403 → Client shows access denied
* **Network Errors**: Client handles connection issues → Shows retry options

### Data Synchronization

* **Session Management**: Client stores user session → Server validates on each request
* **State Management**: Client maintains local state → Server provides authoritative data
* **Cache Strategy**: Client caches group data → Server provides fresh data on updates