

Design document

Troupers (Regular Users)

These stories reflect the core experience of community-driven messaging and discovery.

Core Actions

- **Join and participate in troupes**
- **Send direct messages**
- **Discover communities by interest**
- **Manage their own profile**

As a Trouper, I want to create a new troupe, so that I can start a community around my interests.

As a Trouper, I want to join existing troupes, so that I can connect with like-minded learners.

As a Trouper, I want to send direct messages to other users, so that I can collaborate privately.

As a Trouper, I want to view all my conversations, so that I can continue chats easily.

As a Trouper, I want to update my profile and avatar, so that others can recognize me.

As a Trouper, I want to search for troupes by interest tags, so that I can find relevant communities.

As a Trouper, I want to leave a troupe, so that I can manage my memberships.

As a Trouper, I want to view messages in a troupe, so that I can stay updated on group discussions.

Admins

Admins manage platform health, seed content, and support users. For MVP, their powers can be lightweight but strategic.

Admin Capabilities

- **Oversee user activity**
- **Seed and moderate troupes**
- **Access analytics or logs (optional stretch)**

As an Admin, I want to view all registered users, so that I can monitor platform growth.

As an Admin, I want to delete inappropriate messages, so that I can maintain a safe environment.

As an Admin, I want to create featured troupes, so that I can guide new users toward active communities.

As an Admin, I want to assign admin roles to trusted users, so that I can delegate moderation.

As an Admin, I want to view all conversations and troupe memberships, so that I can audit activity if needed.

As an Admin, I want to remove inactive or abusive users, so that I can protect the community.

As an Admin, I want to tag troupes with interests, so that discovery is easier for new users.

Models Set (Ordered for Migration)

1. User

```
User {  
  id: UUID (PK)  
  name: string  
  email: string (unique)  
  password: string  
  avatar_url: string (nullable)  
  role: enum('admin', 'trouper') // 'trouper' = regular user  
  created_at: timestamp  
}
```

2. InterestTag (*Optional Stretch Goal*)

```
InterestTag {  
  id: UUID (PK)  
  name: string  
}
```

3. Troupe

```
Troupe {  
  id: UUID (PK)  
  name: string  
  description: text  
  visibility: enum('public', 'private')  
  created_by: UUID (FK → User.id)  
  created_at: timestamp  
}
```

4. Conversation

```
Conversation {  
  id: UUID (PK)  
  created_by: UUID (FK → User.id)  
  is_group: boolean (default: false)  
  created_at: timestamp  
}
```

5. ConversationParticipant

```
ConversationParticipant {  
  id: UUID (PK)  
  conversation_id: UUID (FK → Conversation.id)
```

```
  user_id: UUID (FK → User.id)
  joined_at: timestamp
}
```

6. Membership

```
Membership {
  id: UUID (PK)
  user_id: UUID (FK → User.id)
  troupe_id: UUID (FK → Troupe.id)
  joined_at: timestamp
}
```

7. Message (Unified for troupe and conversation)

```
Message {
  id: UUID (PK)
  sender_id: UUID (FK → User.id)
  content: text
  troupe_id: UUID (nullable, FK → Troupe.id)
  conversation_id: UUID (nullable, FK → Conversation.id)
  created_at: timestamp
}
```

💡 **Controller logic tip:** You can enforce that either `troupe_id` or `conversation_id` is present, but not both, using validation rules or DB constraints.

8. UserInterest (Optional Stretch Goal)

```
UserInterest {  
  
  id: UUID (PK)  
  
  user_id: UUID (FK → User.id)  
  
  interest_tag_id: UUID (FK → InterestTag.id)  
  
}
```

Perfect timing, Michael. With your models locked in and user stories defined, controller planning is the next strategic move. Here's a collaborative, sprint-friendly plan for building each controller—complete with purpose, key actions, and suggested team roles.

Controller Development Plan

Each controller below includes:

- **Purpose:** What it manages
 - **Core Actions:** CRUD + custom logic
 - **Team Notes:** Suggestions for delegation or pairing
-

1. AuthController

🔑 **Purpose:** Handles registration, login, logout

🔧 **Core Actions:**

- `register()`: Create user with role `trouper` by default
- `login()`: Issue token/session

- `logout()`: Invalidate token/session
- `me()`: Return authenticated user


 **Team Notes:** Pair backend dev with someone testing frontend auth flows

2. UserController


 **Purpose:** Profile management and role-based access

 **Core Actions:**

- `updateProfile()`: Avatar, bio, etc.
- `getUser(id)`: View user profile
- `listUsers()`: Admin-only view of all users

 **Team Notes:** Assign to someone handling admin dashboard logic

3. TroupeController

 **Purpose:** Create and manage troupes

 **Core Actions:**

- `createTroupe()`
- `getTroupe(id)`
- `listTroupes()`
- `updateTroupe(id)`
- `deleteTroupe(id)` (*admin or creator only*)

 **Team Notes:** Pair with frontend dev working on troupe discovery and creation UI

4. MembershipController

 **Purpose:** Join/leave troupes

 **Core Actions:**

- `joinTroupe(troupe_id)`
- `leaveTroupe(troupe_id)`
- `listMembers(troupe_id)`


 **Team Notes:** Useful for rendering member lists and access control

5. ConversationController

 **Purpose:** Create and manage direct/group conversations

 **Core Actions:**

- `startConversation(user_ids[])`
- `getConversation(id)`
- `listConversations()`

 **Team Notes:** Assign to someone focused on private messaging flows

6. ConversationParticipantController


 **Purpose:** Manage participants in conversations

 **Core Actions:**

- `addParticipant(conversation_id, user_id)`
- `removeParticipant(conversation_id, user_id)`
- `listParticipants(conversation_id)`

 **Team Notes:** Optional for MVP if you only support 1-on-1 chats

7. MessageController


 **Purpose:** Send and retrieve messages

 **Core Actions:**

- `sendMessage()`: Accepts either `troupe_id` or `conversation_id`
- `getMessages(troupe_id | conversation_id)`
- `deleteMessage(id)` (*admin or sender only*)


 **Team Notes:** Pair with frontend dev working on chat UI

8. InterestController (*Stretch Goal*)

 **Purpose:** Manage interest tags

 **Core Actions:**

- `listTags()`
- `assignTagToUser(user_id, tag_id)`
- `getUserTags(user_id)`

 **Team Notes:** Assign if time allows—great for discovery features



Suggested Sprint Allocation

Sprint	Focus Area	Controllers
1	Auth + Troupe Core	AuthController, TroupeController, MembershipController
2	Messaging	ConversationController, MessageController
3	Polish + Admin	UserController, InterestController, optional moderation logic
