MMI — Master Documentation

Belong IRL (MMI) is a web application that connects **clients** with **community mentors** (service providers) within **organizations**. Organization admins manage clients and mentors, create connections between them, and maintain internal ratings of mentors. Mentors see only their own connections and profile. A platform-level **superadmin** manages all organizations.

This document describes the application's purpose, views, technical architecture, API, and each major feature.

Table of contents

---

1. Purpose and scope

- **Goal:** Enable organizations to match their clients with community mentors (helpers), track client–mentor connections, and keep internal notes/ratings about mentors without exposing those to the mentors.
- **Audience:**
  - **Superadmin:** MMI platform team; manages organizations.
  - **Org admin:** Organization staff; manages clients, mentors, and connections for their org.
  - **Service provider (mentor):** Community volunteer; sees only their assigned clients and can update connection status and add session updates.
- **Out of scope (current version):** Sign-up flow, password auth (login is username-only), persistence beyond in-memory store, and email/notifications.

---

2. User roles and capabilities

Role	Description	Key capabilities
superadmin	MMI platform administrator	List/create/update orgs; view any org's dashboard, users, clients, connections; set helper ratings in any org.
orgadmin	Organization administrator	View/manage own org's clients, mentors, connections; create connections; set internal helper ratings; add users (orgadmin or serviceprovider) to the org.
serviceprovider	Community mentor / helper	View own profile and connections; accept/decline pending connections; update connection status (e.g. pause/complete); add session/engagement updates. **Cannot** see internal ratings or org-level lists.

- **Multi-org helpers:** A service provider can belong to multiple organizations (`orgIds`). They choose which org context to work in when logging in (if more than one). Connections and internal ratings are **per-organization**.

---

3. Views and navigation

3.1 Entry and routing

- **Login:** `GET /` (index) → login form; submit posts to `POST /api/auth/login`. On success, user is redirected to `/app.html#<page>` based on role.
- **App shell:** `GET /app.html` is the single-page app. Hash-based routing: `#orgs`, `#dashboard`, `#helper-home`, etc. Only views allowed for the current role are shown; nav links are filtered by role.

3.2 Superadmin views

- **Organizations (`#orgs`):** List of all orgs; "Add organization" opens a modal. Clicking an org goes to org detail.
- **Org detail (`#org-detail`):** Single org view with metrics (clients, helpers, connections, active/pending) and main contacts; "Back to organizations" returns to list.
- **Settings (`#settings`):** Superadmin-specific settings (e.g. display name, username shown).

3.3 Org admin views

- **Home (`#home`):** Landing with link to Dashboard and Figma design.
- **Dashboard (`#dashboard`):** Main workspace with three tabs:
  - **Clients:** List of client cards; "Add client"; click card → client detail modal (profile, suggested mentors, connect, remove client).
  - **Community Mentors:** List of helper cards; "Add Mentor"; each card shows internal rating (stars + notes) and "Add rating" / "Edit rating"; connect dropdown to create connections; click card → mentor detail modal.
  - **Connections:** List of all org connections with filters (client, mentor, status); click row → connection detail (updates, status actions).
- **Settings (`#settings`):** Org-admin settings, Admin (add user to org), Profile.

3.4 Service provider (mentor) views

- **Choose organization (`#helper-choose-org`):** Shown when helper has multiple orgs; pick which org to work in (stored in session).
- **My clients (`#helper-home`):** List of the helper's connections (active/pending) for the chosen org; accept/decline pending; open connection detail.
- **Profile (`#helper-profile`):** Edit display name, bio, "Can mentor in" (needs). Saved via `PATCH /api/users/me`.

---

4. Technical overview

4.1 Stack

- **Backend:** NestJS (Node.js), TypeScript. Serves API under `/api` and static files from `/public` (login page, app shell, JS, CSS).
- **Frontend:** Vanilla JS, no framework. Hash routing, `sessionStorage` for token and user, `fetch` for API with `Authorization: Bearer <token>`.
- **Data:** In-memory store (no database). Restart clears all data except seed (see §12).
- **Validation:** `class-validator` + Nest `ValidationPipe` (whitelist, transform) on request bodies.

4.2 High-level architecture

```
┌─────────────────────────────────────────────────────────────┐
│  Browser: / (login)  or  /app.html#<page>                   │
│  sessionStorage: token, user, (helper) chosen org            │
└──────────────────────────────┬──────────────────────────────┘
                               │
                               ▼
```

```
┌──────────────────────────────────────────────────────────────────┐
│  NestJS server (port 3000)                                       │
│  ├── Static: / → public/index.html, /app.html, /css, /js         │
│  └── API: /api/* → AuthGuard → Controllers → StoreService        │
└──────────────────────────────────────────────────────────────────┘
                                 │
                                 │
                                 ▼
┌──────────────────────────────────────────────────────────────────┐
│  StoreService (in-memory)                                       │
│  users, orgs, clients, connections, connectionUpdates,          │
│  helperRatings, tokenToUserId                                   │
└──────────────────────────────────────────────────────────────────┘
```

4.3 Key backend modules

- **AuthModule:** Login, token validation, `AuthGuard` injects user into request.
- **StoreModule:** Global; `StoreService` holds all entities and auth tokens.
- **OrgsModule, ClientsModule, ConnectionsModule, UsersModule, HelperRatingsModule:** REST resources; all protected by `AuthGuard`; role checks inside controllers.

---

5. API implementation

5.1 Base URL and auth

- **Base URL:** `http://localhost:3000/api` (or deployed origin + `/api`).
- **Auth:** All endpoints except `POST /api/auth/login` and optionally `GET /api` and `GET /api/health` require:
  - Header: `Authorization: Bearer <token>`
  - Token is obtained from login and stored in session; validated by `AuthGuard`; `req.user` is set to the current user (e.g. `UserResponse`).

5.2 Global prefix and validation

- Global prefix: `api` (e.g. `GET /api/health`).
- Global `ValidationPipe`: `whitelist: true`, `forbidNonWhitelisted: true`, `transform: true` (including implicit type conversion for DTOs).

5.3 API summary (by area)

Method	Path	Who	Description
App / health			
GET	`/api`	—	App info (name, version, docs link).
GET	`/api/health`	—	Health (status, timestamp, uptime).
Auth			
POST	`/auth/login`	—	Login by username; returns `{ user, token }`.
GET	`/auth/me`	any	Validate token; returns `{ user }`.
Orgs			
GET	`/orgs`	superadmin	List all orgs.

POST	`/orgs`	superadmin	Create org.
GET	`/orgs/:id`	superadmin, orgadmin (own)	Org detail + metrics + mainContacts.
PATCH	`/orgs/:id`	superadmin	Update org.
Clients			
GET	`/orgs/:orgId/clients`	superadmin, orgadmin (own)	List clients of org.
POST	`/orgs/:orgId/clients`	superadmin, orgadmin (own)	Create client.
Users			
GET	`/orgs/:orgId/users`	superadmin, orgadmin (own)	List users in org (includes `internalRating` for serviceproviders).
POST	`/orgs/:orgId/users`	superadmin, orgadmin (own)	Add user (orgadmin or serviceprovider) to org.
GET	`/users/me`	any	Current user profile.
PATCH	`/users/me`	any	Update own profile (displayName, bio, needs).
Connections			
GET	`/connections/me`	serviceprovider	My connections (optional `?orgId=`).
GET	`/connections/:id`	helper of connection, or orgadmin/superadmin (org)	One connection + client + helper + updates.
POST	`/connections/:id/updates`	helper of connection	Add session/engagement update.
PATCH	`/connections/:id/status`	helper or orgadmin/superadmin	Set status (e.g. active, paused, complete).
PATCH	`/connections/:id/accept`	serviceprovider (own)	Accept pending connection.
PATCH	`/connections/:id/decline`	serviceprovider (own)	Decline pending connection.
GET	`/orgs/:orgId/connections`	superadmin, orgadmin (own)	List org connections + client + helper + updates + internalRating.
POST	`/orgs/:orgId/connections`	orgadmin, superadmin	Create connection (pending until helper accepts).
Helper ratings			

GET	`/orgs/:orgId/helpers/:helperId/rating`	superadmin, orgadmin (own)	Get internal rating (stars, notes).
PUT	`/orgs/:orgId/helpers/:helperId/rating`	superadmin, orgadmin (own)	Set internal rating (1–5 stars, optional notes).

---

6. Feature: Authentication

6.1 Purpose

Identify the user by username and issue a session token so subsequent API calls can be authorized by role and org.

6.2 Flow

1. User submits username on login page → `POST /api/auth/login` with `{ username }`.
2. Backend: `AuthService.login(username)` looks up user by username, creates a token, stores `token → userId` in store, returns `{ user, token }`. User is serialized as `UserResponse` (no internal-only fields).
3. Frontend stores `token` and `user` in sessionStorage and redirects to `/app.html#<page>` (page chosen by `userType` and, for serviceprovider, org count).
4. Every protected request sends `Authorization: Bearer <token>`.
5. `AuthGuard` resolves token to user; if invalid/missing, returns 401.

6.3 Implementation notes

- **AuthController:** `POST auth/login`, `GET auth/me`. No auth guard on login; `auth/me` uses header only (no guard in the snippet but typically used to re-validate).
- **AuthGuard:** Reads `Authorization`, extracts Bearer token, calls `AuthService.validateToken(token)`; attaches `user` to request.
- **Login DTO:** `username` (string, required). No password in current design.

---

7. Feature: Organizations

7.1 Purpose

Organizations are the top-level tenant. Each org has a name, main contact name/email, and is the container for clients, users (orgadmin + serviceproviders), and connections. Superadmin can create and edit orgs; orgadmin can only view their own org.

7.2 API

- **GET /api/orgs** — List all orgs (superadmin only).
- **POST /api/orgs** — Create org (superadmin only). Body: `CreateOrgDto` (name, mainContactName, mainContactEmail).
- **GET /api/orgs/:id** — Org by id; caller must be superadmin or orgadmin for that org. Response includes `metrics` (clientsCount, helpersCount, connectionsCount, activeConnections, pendingConnections) and `mainContacts`.
- **PATCH /api/orgs/:id** — Update org (superadmin only). Body: optional name, mainContactName, mainContactEmail.

7.3 Frontend

- Superadmin: Organizations list and org detail pages; add-org modal with form. Org cards link to `#org-detail` with org id in state/hash.

---

8. Feature: Users and org membership

8.1 Purpose

Users belong to one or more orgs and have a role (superadmin, orgadmin, serviceprovider). Org admins can add users (orgadmin or serviceprovider) to their org. Service providers can belong to multiple orgs (`orgIds`). Listing org users returns `internalRating` for each serviceprovider (for org admin UI only; see §11).

8.2 API

- **GET /api/orgs/:orgId/users** — List users in org (superadmin or orgadmin for that org). Each user is returned as `UserResponse` ; for `userType === 'serviceprovider'` , includes `internalRating: { stars, notes }` when present.
- **POST /api/orgs/:orgId/users** — Add user to org (superadmin or orgadmin). Body: `AddUserDto` (username, userType: orgadmin | serviceprovider, displayName, optional bio, needs). Username must be unique globally.
- **GET /api/users/me** — Current user profile (any authenticated user).
- **PATCH /api/users/me** — Update own profile. Body: `UpdateProfileDto` (displayName, bio, needs). Used by service providers to edit their mentor profile.

8.3 Implementation notes

- User lookup: by username (unique); store assigns id and createdAt. Service providers with multiple orgs have `orgIds` ; single-org users use `orgId` (and may have `orgIds` for consistency).
- `UserResponse` includes `orgNames` (resolved from org ids) for display (e.g. badges).

---

9. Feature: Clients

9.1 Purpose

Clients are people that an organization supports. They are matched with community mentors (service providers) via connections. Only the owning org (and superadmin) can see and manage clients.

9.2 API

- **GET /api/orgs/:orgId/clients** — List clients for org (superadmin or orgadmin for that org).
- **POST /api/orgs/:orgId/clients** — Create client (superadmin or orgadmin). Body: `CreateClientDto` (name, optional age, bio, needs). Client is created with `orgId` and server-generated id and createdAt.

9.3 Data model

- **Client:** id, orgId, name, age?, bio?, needs?, and extended fields (address, contact, charge, atiPlan, story, notes, advocateContact), createdAt.
- **Store:** `deleteClient(id)` exists but is not exposed via a DELETE endpoint in the current ClientsController (used internally or for future use).

9.4 Frontend

- Dashboard → Clients tab: client cards, “Add client” modal, client detail modal (profile, suggested mentors, connect flow, remove client). Remove client calls a delete API if/when implemented; otherwise only store is aware of delete.

---

10. Feature: Connections

10.1 Purpose

A **connection** links one client and one community mentor (helper) within an org. It represents a mentoring relationship. An org admin creates a connection (status `pending`); the helper accepts or declines. Once accepted, status becomes `active` ; it can later be set to `paused` or `complete` . Helpers can add **connection updates** (session/engagement notes and optional media). Connections are org-scoped; the same helper can have different connections in different orgs.

10.2 Connection lifecycle

- **pending** — Created by org admin; waiting for helper to accept or decline.
- **active** — Helper accepted; mentoring relationship active.

- **paused** — Temporarily paused (e.g. break).
- **complete** — Relationship ended.
- **declined** — Helper declined the request.

10.3 API

- **GET /api/connections/me** — Service provider only. Returns connections where the current user is the helper; optional query `orgId` to filter by org. Each item includes connection + client.
- **GET /api/connections/:id** — Single connection by id. Caller must be the helper of that connection, or superadmin/orgadmin with access to the connection's org. Response: connection + client + helper + updates (with createdByDisplayName).
- **POST /api/connections/:id/updates** — Add an update (helper only, for their connection). Body: eventName, eventTime, notes?, media?.
- **PATCH /api/connections/:id/status** — Set status (active, paused, complete). Helper can update own connection; orgadmin/superadmin can update any connection in their org(s).
- **PATCH /api/connections/:id/accept** — Service provider only; accept a pending connection (sets status to active, acceptedAt).
- **PATCH /api/connections/:id/decline** — Service provider only; decline a pending connection (sets status to declined, declinedAt).
- **GET /api/orgs/:orgId/connections** — List all connections for org (superadmin or orgadmin). Each item includes connection, client, helper (with `internalRating` when present), and updates.
- **POST /api/orgs/:orgId/connections** — Create connection (orgadmin or superadmin). Body: `clientId`, `helperId`. Helper must be a serviceprovider in that org (`orgId` or `orgIds` includes org). New connection has status `pending` and `createdById` set to current user.

10.4 DTOs and types

- **Connection:** id, orgId, clientId, helperId, status, createdById, createdAt, acceptedAt?, declinedAt?.
- **ConnectionUpdate:** id, connectionId, eventName, eventTime, notes?, media?, createdBy (helperId), createdAt. Media items: url, type (video | image | audio).
- **CreateConnectionDto:** clientId, helperId. **PatchConnectionStatusDto:** status (active | paused | complete). **CreateConnectionUpdateDto:** eventName, eventTime, notes?, media?.

10.5 Frontend

- Dashboard → Connections tab: table/list with filters (client, mentor, status); click row → connection detail modal (updates, status actions).
- Dashboard → Clients: from client card, "Connect" → choose mentor → confirm → create connection.
- Dashboard → Community Mentors: from helper card, "Connect" dropdown → choose client → confirm.
- Service provider: "My clients" lists connections; accept/decline; open connection detail; add updates; change status (pause/complete).

---

11. Feature: Helper (internal) ratings

11.1 Purpose

Allow org admins (and superadmin) to record an **internal** 1–5 star rating and optional notes for each community mentor (service provider) **per organization**. These ratings are for internal use only: the mentor **never** sees them (not in profile, not in connection responses, and not in any endpoint the mentor can call).

11.2 Design principles

- **Org-specific:** One org can rate a helper 3 stars, another 5 stars (same helper, different `orgId`).
- **Visibility:** Only returned in endpoints that only orgadmin/superadmin call: `GET /api/orgs/:orgId/users` , `GET /api/orgs/:orgId/connections` (on the helper object), and `GET /api/orgs/:orgId/helpers/:helperId/rating` . Service providers do not call these; they only use `users/me` , `connections/me` , and `connections/:id` (no rating attached).

11.3 API

- **GET /api/orgs/:orgId/helpers/:helperId/rating** — Get the org's rating for that helper (superadmin or orgadmin for that org). Returns `{ stars, notes }` or `{ stars: null, notes: null }` if none. Helper must be a serviceprovider in that org.
- **PUT /api/orgs/:orgId/helpers/:helperId/rating** — Set rating (orgadmin or superadmin only). Body: `SetHelperRatingDto` : `stars` (1–5, integer), optional `notes` (string, max 2000 chars). Creates or updates the single rating per (orgId, helperId). Returns `{ stars, notes }`.

11.4 Data model

- **HelperRating:** id, orgId, helperId, stars (1–5), notes?, createdById, createdAt, updatedAt. One rating per (orgId, helperId); upsert on PUT.

11.5 Frontend

- Dashboard → Community Mentors: each helper card shows "Internal rating (not visible to mentor)" with star display (e.g. ★★★★☆), optional notes snippet, and "Add rating" / "Edit rating" button. Clicking opens a modal: 5 star buttons, notes textarea, Save/Cancel. Save calls `PUT .../rating` and refreshes the card and in-memory helpers list.

---

12. Data model and store

12.1 Entities (types)

- **User:** id, username, userType, orgId, orgIds?, displayName, bio?, needs?, createdAt.
- **Org:** id, name, mainContactName, mainContactEmail, createdAt.
- **Client:** id, orgId, name, age?, bio?, needs?, address?, contact?, charge?, atiPlan?, story?, notes?, advocateContact?, createdAt.
- **Connection:** id, orgId, clientId, helperId, status, createdById, createdAt, acceptedAt?, declinedAt?.
- **ConnectionUpdate:** id, connectionId, eventName, eventTime, notes?, media?, createdBy, createdAt.
- **HelperRating:** id, orgId, helperId, stars, notes?, createdById, createdAt, updatedAt.

12.2 StoreService (in-memory)

- **Maps:** users, orgs, clients, connections, connectionUpdates, helperRatings; tokenToUserId (token → userId).
- **Seeding:** On startup, seed creates one superadmin, two orgs, two service providers (one in both orgs, one in first org only), org admin, sample clients, sample connections, and sample helper ratings.
- **IDs:** UUIDs generated in store. No persistence; restart resets everything to seed state.

12.3 Access patterns

- Users by id, by username; users by org (orgId or orgIds).
- Orgs by id; all orgs.
- Clients by id, by orgId.
- Connections by id, by orgId, by helperId.
- Updates by connectionId.
- Helper ratings: get/set by (orgId, helperId).

---

13. Frontend architecture

13.1 Pages and routing

- **Pages** are divs with id `page-<name>` ; visibility is toggled by hash (`#orgs` , `#dashboard` , etc.). Only one page visible at a time. Role determines allowed pages and default page after login.
- **Navigation:** Sidebar and header links use `data-page` and `data-role` ; script shows/hides links by current user role and sets active state.

13.2 Dashboard tabs (org admin)

- Tabs: Clients, Community Mentors, Connections. Each tab shows a panel (`cc-panel-clients` , `cc-panel-helpers` , `cc-panel-connections`). Tab click switches active tab and visible panel; counts in tab labels are updated from loaded data.

13.3 Data loading

- **Dashboard:** When org admin lands on dashboard, `loadDashboardData()` runs: fetches org, clients, users, connections for current user's orgId; filters users to helpers; sets `__mmiDashboardHelpers` , `__mmiDashboardClients` ; binds client/helper ids to cards; updates counts and connection suggestions; renders internal rating blocks on helper cards.
- **Service provider:** "My clients" and profile load from `GET /api/connections/me` and `GET /api/users/me` ; profile save uses `PATCH /api/users/me` .

13.4 Modals

- Add client, add mentor, add org, add user; client detail, helper detail, connection detail, event detail; connect confirm; add update (session); internal rating edit; settings; admin; profile. Modals are shown/hidden by toggling `hidden` and populated from current selection/state.

13.5 Auth and API helper

- Token and user read from sessionStorage. `apiRequest(url, options)` adds `Authorization: Bearer <token>` and handles 401 (redirect to login). Used for all authenticated requests.

Document info

- **Version:** 1.0
- **Last updated:** 2025
- **Codebase:** MMI (Belong IRL); NestJS backend, vanilla JS frontend, in-memory store.

For setup and scripts, see README.md. For design reference, see FIGMA.md and the Figma design link in the README.