Permission System

Synthetic Users

Version: 1.1

Effective Date: March 25, 2026

Last Updated: March 25, 2026

Document Owner: CTO — Artur Ventura

Classification: Internal – Confidential

CRA Reference: 9.1.2

Change History

Version	Date	Author	Changes
1.0	Prior	Engineering Team	Initial stub
1.1	March 25, 2026	Artur Ventura, CTO	Full document written from source code review. Covers two-tier RBAC model, all authentication methods, role definitions, permission matrices, automatic role assignment rules, API enforcement, workspace API keys, and audit logging. Per JPMC SCA CRA 9.1.2.

---

1. Overview

Synthetic Users implements a two-tier Role-Based Access Control (RBAC) model enforced at the API layer. Access is controlled at two scopes:

1. **Workspace scope** — controls which workspace a user belongs to and whether they are an owner or member.
2. **Project scope** — controls which projects within a workspace a user can access and what they can do (admin, editor, or viewer).

Every API request is authenticated before authorization checks are applied. Authorization is enforced in service-layer logic before any data is returned or mutated. Unauthorized requests receive HTTP 401 or 403 responses with machine-readable error codes.

---

2. Authentication Methods

Synthetic Users supports five authentication methods. Each produces a `UserSession` object that subsequent authorization checks operate on.

Method	Token Format	Scope	Used By
Firebase JWT	Bearer JWT, signed by Firebase	User identity	Web app (primary)
Workspace API Key	`su_ws_` prefix, HMAC-hashed in DB	Workspace-scoped	Integrations, external callers
User API Token	Opaque token stored on `User` record	User identity (no workspace scope)	SDK, internal tooling
WebSocket Token	Query param `?k=<token>` (user API token)	User identity	WebSocket connections
Shared Read Key	Fixed shared key	Read-only, limited endpoints	Public share links

2.1 Firebase JWT (Primary)

The web application authenticates users via Firebase Authentication. The client presents a signed JWT in the `Authorization: Bearer <token>` header. The backend decodes the JWT (without verifying the signature in dev mode; with full verification in production via Firebase project ID). The `user_id` claim is used to look up the `User` record.

Anonymous sessions are supported via Firebase Anonymous Auth. Anonymous users are identified by `provider_id == 'anonymous'` and are subject to usage limits.

2.2 Workspace API Keys

Workspace API keys allow programmatic access scoped to a single workspace. Keys are generated with the `su_ws_` prefix and a 50-byte URL-safe random suffix. Only a hashed version (`key_hash`) is stored in the database — the plaintext key is shown to the user once at creation.

Key properties:

- Tied to a specific `workspace_id` and the `user_id` of the creator
- Support optional expiration (`expiration_date`)
- Can be deactivated (`is_active = false`)
- `last_used_at` is tracked for audit purposes
- Indexed on `(workspace_id, is_active)` and `(user_id, workspace_id)`

When authenticated via a workspace API key, the resulting `UserSession` has `workspace_id` set, scoping all operations to that workspace.

2.3 User API Token

Each `User` record has an opaque `api_token` field. This token provides user-level authentication without workspace scoping. Used by the SDK and internal tooling.

2.4 WebSocket Authentication

WebSocket connections authenticate via a `?k=<token>` query parameter (user API token). A dedicated authentication path acquires a database session, verifies the token, and immediately releases the session to avoid holding connections open for the duration of the WebSocket session.

2.5 Shared Read Key

A fixed shared key provides read-only GET access to a limited set of public endpoints (studies, audiences, problems, solutions, synthetic users, interviews, summaries, research goals, knowledge graphs, plans, files). This is used to power shareable read-only links. Any request using the shared key that is not a GET to an allowlisted endpoint is rejected.

---

3. Workspace-Level Roles

Workspace membership is stored in the `users_workspaces` table with a unique constraint on `(user_id, workspace_id)`.

3.1 Workspace Roles

Role	Description
owner	Full control over the workspace. Can manage members, change roles, delete the workspace, manage billing, toggle workspace settings (PII filter, allowed domain), and manage all projects. Owners are automatically added to all projects with the `admin` project role.
member	Standard workspace member. Can access only the projects they have been explicitly added to. Cannot manage workspace settings, billing, or other members.

3.2 Workspace Membership Status

Status	Description
active	User is an active member of the workspace.
pending	User has been invited but has not yet accepted. Pending users appear in workspace member lists but cannot access workspace data until they accept.
inactive	User has been removed from the workspace. Inactive records are excluded from all workspace queries via `status != 'inactive'` filter.

3.3 Workspace-Owner-Only Operations

The following operations require the requesting user to hold the `owner` role in the workspace (`WORKSPACE_OWNER_REQUIRED` error returned otherwise):

- Update workspace description
- Toggle PII filter
- Delete workspace
- Remove a user from the workspace
- Grant or revoke the `owner` role for another user
- Bulk update a member's workspace role and project permissions (`update_member_access`)
- Manage workspace API keys

---

4. Project-Level Roles

Project membership is stored in the `users_projects` table with a unique constraint on `(user_id, project_id)` .

4.1 Project Roles

Role	Description

admin	Full control over the project. Can manage project members, configure project settings, run studies, view all data, and delete the project.
editor	Can create and edit content within the project (synthetic users, studies, problems, solutions, interviews) but cannot manage project membership or settings.
viewer	Read-only access to project data. Cannot create, edit, or delete content.

4.2 Project Membership Status

Status	Description
active	User is an active project member.
pending	User has been invited to the project but has not yet accepted.
inactive	User has been removed from the project. Excluded from project queries via `status != 'inactive'` filter.

4.3 Project-Admin-Only Operations

The following operations require the requesting user to hold the `admin` role in the project (`PROJECT_ADMIN_REQUIRED` error returned otherwise):

- Invite or remove project members
- Update project settings
- Delete the project

---

5. Permission Matrix

5.1 Workspace Operations

Operation	Owner	Member
View workspace	☐	☐

View workspace members	✓	✓
View workspace usage stats	✓	✓
Update workspace description	✓	✗
Toggle PII filter	✓	✗
Delete workspace	✓	✗
Add / remove members	✓	✗
Grant / revoke owner role	✓	✗
Manage workspace API keys	✓	✗
View projects	✓ (all)	✓ (assigned only)
Create project	✓	✗

5.2 Project Operations

Operation	Workspace Owner	Project Admin	Project Editor	Project Viewer
View project	✓	✓	✓	✓
View studies, interviews, outputs	✓	✓	✓	✓
Create / edit synthetic users	✓	✓	✓	✗
Run studies / interviews	✓	✓	✓	✗
Upload files	✓	✓	✓	✗
Manage project members	✓	✓	✗	✗
Update project settings	✓	✓	✗	✗
Delete project	✓	✓	✗	✗

6. Automatic Role Assignment

The following role assignments happen automatically without manual configuration:

6.1 New Workspace

When a new user logs in for the first time and has no workspace, the system creates a default workspace and assigns the user the `owner` role.

6.2 New Project

When a user creates a new project:

- The **creator** is automatically added to the project with the `admin` role and `active` status.
- All current **workspace owners** are automatically added to the project with the `admin` role and `active` status.

This ensures workspace owners always have full visibility and control over all projects in their workspace without needing to be manually added.

6.3 Granting Workspace Owner Role

When a member is promoted to workspace `owner` via the `update_member_access` endpoint:

- Their workspace role is updated to `owner`.
- They are immediately granted `admin` project access to **all** projects in the workspace.

6.4 Revoking Workspace Owner Role

When an owner is demoted to `member`:

- Their workspace role is updated to `member`.

- Their project access is synced to match the explicit `project_permissions` provided in the request. Any projects not listed are removed.

6.5 Joining a Workspace via Allowed Domain

If a workspace has an `allowed_domain` configured, users with a matching email domain who log in are automatically added to the workspace with the `member` role and `active` status.

---

7. Invitation Flow

7.1 Workspace Invitations

1. A workspace owner sends an invitation to an email address (stored as a `WorkspaceInvite` with `status = 'pending'`).
2. The invited user is visible in the workspace member list with `pending` status.
3. On acceptance, a `UsersWorkspaces` record is created with `status = 'active'` and the configured role.
4. The invitation `status` is updated to a terminal state.

Seats limit is checked before sending invitations. If `workspace.total_users_and_invited_users >= subscription.seats_limit`, the invitation is rejected with `WORKSPACE_SEATS_LIMIT_REACHED`.

7.2 Project Invitations

1. A project admin invites a user by email (stored as a `ProjectInvite` with `status = 'pending'`).
2. On acceptance, a `UsersProjects` record is created with `status = 'active'` and the assigned role.
3. The invitee must already be a member of the parent workspace.

---

8. API Enforcement

Authorization is enforced in the service layer before any data access or mutation. The enforcement pattern is:

```
1. Authenticate request → UserSession
2. Load resource (workspace or project)
3. Check user membership in resource
4. Check role requirement (owner / admin as needed)
5. Execute operation or raise HTTP exception
```

Error codes returned on authorization failure:

Error Code	HTTP Status	Condition
`user_unauthorized`	401	User is not a member of the project
`workspace_owner_required`	403	Operation requires workspace owner role
`project_admin_required`	403	Operation requires project admin role
`not_authenticated`	403	No valid token present
`invalid_token`	403	Token present but invalid
`email_not_verified`	403	User email not verified
`workspace_not_found`	404	Workspace does not exist or is deleted
`workspace_user_not_found`	404	User is not in the workspace

9. Data Isolation

9.1 Workspace Isolation

Each workspace is an independent data container. Users can belong to multiple workspaces but cannot access data across workspace boundaries. All queries are filtered by `workspace_id`.

9.2 Project Isolation within a Workspace

Within a workspace, projects are isolated from each other. Members only see projects they are explicitly added to. The `get_workspace_projects` method filters the workspace's project list to return only those where the requesting user has a `UsersProjects` record with `status != 'inactive'`.

9.3 Soft Deletion

Workspaces and projects are soft-deleted (`deleted = true`). All queries include `deleted == false` filters. Soft-deleted resources are invisible to all users.

9.4 PII Filtering

Workspaces can enable a PII filter (`pii_filter_enabled`). When enabled, personally identifiable information in research outputs is filtered before being returned. Only workspace owners can toggle this setting.

---

10. Audit Logging

All significant permission and access events are logged via the `log_audit_event` service. Logged events include:

Event	Logged Fields

`create_workspace`	workspace_id, workspace_description
`update_workspace_description`	workspace_id, new_description
`delete_workspace`	workspace_id
`remove_user_from_workspace`	workspace_id, target_user_id
`grant_owner_role`	workspace_id, target_user_id
`revoke_owner_role`	workspace_id, target_user_id
`update_member_access`	workspace_id, target_user_id, new role and project permissions
`create_project`	project_id, workspace_id
`delete_project`	project_id
`invite_member`	workspace or project, invitee email, role

Audit logs are retained per the Information Governance & Records Management Standard.

---

11. Workspace API Key Lifecycle

Lifecycle Stage	Behaviour
Creation	Generated by workspace owner. Plaintext key shown once; only hash stored.
Active use	`last_used_at` timestamp updated on each successful authentication.
Expiration	Keys with `expiration_date` in the past are treated as invalid.

Deactivation	Owner can set `is_active = false` to immediately revoke a key without deleting the record.
Rotation	Old key is deactivated; new key is created. Rotated on personnel departure per the Access Rights Review Policy.
Deletion	Key record can be permanently deleted by a workspace owner.

---

12. Related Documents

- Access Rights Review Policy — Semi-annual review of role assignments including LLM API credentials
- Change Management Policy — Role or permission configuration changes are governed change events
- Incident Response Plan — Unauthorized access incident procedures
- Password Management Policy — Credential management for API keys and service accounts
- Third-Party Risk Management Policy — API integration access governance