# Architectural Design (AI Directory)

**Entity Design Notes**

1. users

**Purpose**: Core authentication and identity table for all system users (members, admins, moderators).

**Key Fields**:

* username — unique, indexed for fast login.
* password — store a salted hash (e.g., bcrypt), never plaintext.
* email — unique, indexed for account recovery.

**Core Fields:**

* id (PK, SERIAL or BIGSERIAL)
* username (CITEXT, unique)
* password (TEXT, hashed)
* email (CITEXT, unique)
* last\_login (TIMESTAMP WITH TIME ZONE)

**Metadata:**

* created\_at (TIMESTAMP WITH TIME ZONE, default NOW())
* updated\_at (TIMESTAMP WITH TIME ZONE, auto‑update trigger)

**Constraints:**

* Unique on username and email
* CHECK for valid email format (optional)
* Index on last\_login for activity queries

**Postgres Notes**:

* Use citext for case‑insensitive username/email matching.
* Add CHECK constraints for email format if desired.
* Consider last\_login defaulting to NULL until first login.

2. members

**Purpose**: Extends users with profile and membership‑specific data.

**Key Fields**:

* user\_id — FK to users, ON DELETE CASCADE to remove member if user is deleted.
* member\_role\_id — FK to member\_role.
* membership\_status — boolean for active/inactive.

**Core Fields:**

* id (PK)
* user\_id (FK → users.id)
* first\_name (VARCHAR(255))
* last\_name (VARCHAR(255))
* member\_role\_id (FK → member\_role.id)
* email (CITEXT)
* phone (TEXT or NUMERIC)
* website (TEXT)
* last\_update (TIMESTAMP WITH TIME ZONE)
* memberships\_id (FK → memberships.id)
* membership\_status (BOOLEAN)

**Metadata:**

* created\_at, updated\_at

**Constraints:**

* FK ON DELETE CASCADE for user\_id
* Index on member\_role\_id and membership\_status

Optional unique on email if not shared with users

**Postgres Notes**:

* Index member\_role\_id for role‑based queries.
* Use TEXT for phone if you need to store formatting symbols; NUMERIC only if strictly digits.

3. member\_profile

**Purpose**: Rich profile content, media, and visibility controls.

**Key Fields**:

* member\_id — FK to members.
* visibility — boolean; could be extended to enum for finer control.

**Core Fields:**

* id (PK)
* member\_id (FK → members.id)
* profile\_photo (BYTEA or URL)
* background\_photo (BYTEA or URL)
* headline (VARCHAR(255))
* about (TEXT)
* work\_experience (TEXT)
* years\_experience (NUMERIC)
* education (VARCHAR(255))
* interests (VARCHAR(255))
* linkedin (TEXT)
* youtube (TEXT)
* visibility (BOOLEAN)

**Metadata:**

* created\_at, updated\_at

**Constraints:**

* FK ON DELETE CASCADE for member\_id
* Index on visibility
* Optional full‑text index on about and work\_experience

**Postgres Notes**:

* Store images in BYTEA for simplicity, or use TEXT URLs if storing in object storage.
* Add GIN index on about or work\_experience if full‑text search is needed.

4. member\_account

**Purpose**: Tracks subscription linkage and lifecycle dates.

**Key Fields**:

* member\_id — FK to members.
* subscription\_id — FK to subscriptions.

**Core Fields:**

* id (PK)
* member\_id (FK → members.id)
* subscription\_id (FK → subscriptions.id)
* registration\_date (DATE)
* next\_payment\_date (DATE)
* expiration\_date (DATE)

**Metadata:**

* created\_at, updated\_at

**Constraints:**

* Composite unique on (member\_id, subscription\_id)
* Index on expiration\_date for active account queries

**Postgres Notes**:

* Add partial index on expiration\_date for active accounts.
* Use DATE for billing cycles; TIMESTAMP WITH TIME ZONE if time precision matters.

5. subscriptions

**Purpose**: Represents a purchased subscription instance.

**Key Fields**:

* Links to memberships, payments, discounts, billing.

**Core Fields:**

* id (PK)
* purchase\_date (DATE)
* amount (NUMERIC(10,2))
* member\_id (FK → members.id)
* memberships\_id (FK → memberships.id)
* payment\_id (FK → payments.id)
* discounts\_id (FK → discounts.id)
* billing\_id (FK → billing.id)

**Metadata:**

* created\_at, updated\_at

**Constraints:**

* FK ON DELETE SET NULL for optional relationships
* Index on member\_id and purchase\_date

**Postgres Notes**:

* Consider NUMERIC(10,2) for amount to avoid float rounding.
* FK constraints should be ON DELETE SET NULL for optional relationships.

6. memberships

**Purpose**: Defines subscription tiers/plans.

**Key Fields**:

* price — NUMERIC(10,2) for currency.
* allow\_signups — boolean flag.

**Core Fields:**

* id (PK)
* name (VARCHAR(255))
* description (TEXT)
* price (NUMERIC(10,2))
* allow\_signups (BOOLEAN)

**Metadata:**

* None required beyond PK

**Constraints:**

* Unique on name

**Postgres Notes**:

* Add unique index on name to prevent duplicates.

7. payments

**Purpose**: Stores payment transaction references.

**Key Fields**:

* stripe\_charge\_id — unique, indexed.

**Core Fields:**

* id (PK)
* stripe\_charge\_id (VARCHAR(255))

**Metadata:**

* None required beyond PK

**Constraints:**

* Unique on stripe\_charge\_id

**Postgres Notes**:

* Could add JSONB column for gateway metadata.

8. discounts

**Purpose**: Promotional codes and descriptions.

**Key Fields**:

* discount\_code — unique, indexed.

**Core Fields:**

* id (PK)
* discount\_code (VARCHAR(255))
* description (TEXT)

**Metadata:**

* None required beyond PK

**Constraints:**

* Unique on discount\_code

**Postgres Notes**:

* Add CHECK constraint for code format.

9. billing

**Purpose**: Billing entity (person or company).

**Core Fields:**

* id (PK)
* name (VARCHAR(255))

**Metadata:**

* None required beyond PK

**Constraints:**

* None beyond PK

**Postgres Notes**:

* Could be extended with address/contact fields.

10. member\_role

**Purpose**: Role definitions (e.g., standard, premium, admin).

**Core Fields:**

* id (PK)
* name (VARCHAR(255))

**Metadata:**

* None required beyond PK

**Constraints:**

* Unique on name

**Postgres Notes**:

* Keep name unique.

11. skills

**Purpose**: Controlled vocabularies for tagging profiles.

**Core Fields:**

* id (PK)
* name (VARCHAR(255))

**Metadata:**

* None required beyond PK

**Constraints:**

* Unique on name

**Postgres Notes**:

* All should have unique name fields.
* Index name for autocomplete.

12. expertise, industries

**Purpose**: Controlled vocabularies for tagging profiles.

**Core Fields:**

* id (PK)
* name (VARCHAR(255))

**Metadata:**

* None required beyond PK

**Constraints:**

* Unique on name

**Postgres Notes**:

* All should have unique name fields.
* Index name for autocomplete.

13. industries

**Purpose**: Controlled vocabularies for tagging profiles.

**Core Fields:**

* id (PK)
* name (VARCHAR(255))

**Metadata:**

* None required beyond PK

**Constraints:**

* Unique on name

**Postgres Notes**:

* All should have unique name fields.
* Index name for autocomplete.

14. recommendations

**Purpose**: Peer reviews/endorsements with moderation.

**Key Fields**:

* moderation\_status — enum type in Postgres (CREATE TYPE moderation\_status AS ENUM(...)).

**Core Fields:**

* id (PK)
* user\_id (FK → users.id)
* member\_id (FK → members.id)
* rating\_value (INTEGER)
* content (VARCHAR(255))
* date (TIMESTAMP WITH TIME ZONE)
* moderation\_status (moderation\_status ENUM)

**Metadata:**

* None beyond PK

**Constraints:**

* Index on moderation\_status for moderation queues
* FK ON DELETE CASCADE for member\_id

**Postgres Notes**:

* Index moderation\_status for moderation queues.

15. member\_skills

**Purpose**: Join tables for many‑to‑many relationships.

**Core Fields:**

* id (PK)
* member\_profile\_id (FK → member\_profile.id)
* skill\_id (FK → skills.id)

**Metadata:**

* None beyond PK

**Constraints:**

* Composite unique on (member\_profile\_id, skill\_id)

**Postgres Notes**:

* Composite unique index on (member\_profile\_id, skill\_id) etc. to prevent duplicates.

16. member\_expertise

**Purpose**: Join tables for many‑to‑many relationships.

**Core Fields:**

* id (PK)
* member\_profile\_id (FK → member\_profile.id)
* expertise\_id (FK → expertise.id)

**Metadata:**

* None beyond PK

**Constraints:**

* Composite unique on (member\_profile\_id, expertise\_id)

**Postgres Notes**:

* Composite unique index on (member\_profile\_id, skill\_id) etc. to prevent duplicates.

17. member\_industries

**Purpose**: Join tables for many‑to‑many relationships.

**Core Fields:**

* id (PK)
* member\_profile\_id (FK → member\_profile.id)
* industry\_id (FK → industries.id)

**Metadata:**

* None beyond PK

**Constraints:**

* Composite unique on (member\_profile\_id, industry\_id)

**Postgres Notes**:

* Composite unique index on (member\_profile\_id, skill\_id) etc. to prevent duplicates.

18. member\_recommendations

**Purpose**: Links profiles to recommendations.

**Core Fields:**

* id (PK)
* user\_id (FK → users.id)
* member\_id (FK → members.id)
* rating\_value (INTEGER)
* content (VARCHAR(255))
* date (TIMESTAMP WITH TIME ZONE)
* moderation\_status (moderation\_status ENUM)

**Metadata:**

* None beyond PK

**Constraints:**

* Index on moderation\_status for moderation queues
* FK ON DELETE CASCADE for member\_id

**Postgres Notes**:

* Composite unique index on (member\_profile\_id, recommendation\_id).

19. audit\_log

**Purpose**: Immutable record of significant actions for security, compliance, and debugging (e.g., CRUD ops, logins, permission changes).

**Key Fields & FKs**:

* id — PK, auto‑increment.
* user\_id — FK → users.id (nullable if system‑generated).
* action\_type — short descriptor (e.g., UPDATE\_PROFILE).
* entity\_name / entity\_id — target of the action.
* ip\_address — INET type for IPv4/IPv6.
* user\_agent — client metadata.

**Core fields:**

* id (PK)
* user\_id (FK → users.id)
* action\_type (VARCHAR(255)) — short label like CREATE\_MEMBER, DELETE\_SUBSCRIPTION.
* action\_details (TEXT) — free‑form JSON or text for context.
* entity\_name (VARCHAR(255))
* entity\_id (INTEGER)
* ip\_address (INET)
* user\_agent (VARCHAR(255))

**Metadata:**

* ip\_address — store as INET for IPv4/IPv6.
* user\_agent — capture browser/app info.
* created\_at — UTC timestamp of the event.

**Constraints:**

* No updates/deletes — append‑only table.
* No cascading deletes — preserve history
* Index on (entity\_name, entity\_id) for reverse lookups.
* Index on created\_at for time‑range queries.

**Postgres Notes**:

* Partition by date for large volumes.
* Index (entity\_name, entity\_id) for targeted lookups.
* No ON UPDATE or ON DELETE cascades — preserve history.