My Fitness Community - Documentation

- My Fitness Community (لياقتي) — Documentation Package
  - Index
  - Folders
  - Conventions
- 01 — Project Overview
  - Project title
  - Academic context
  - Purpose
  - Problem statement
  - Proposed solution
  - Objectives (from the presentation, reflected on the Home page)
  - Target users
  - Scope
  - Technologies actually used
- 02 — Requirements Analysis
  - Functional requirements
  - Non-functional requirements (supported)
  - Requirements represented in the academic project but not implemented
- 03 — System Roles
  - Visitor (not signed in)
  - Subscriber (مشترك)
  - Specialist / Trainer (مدرب متخصص)
  - Admin (مشرف)
  - Authentication / authorization summary
- 04 — System Interfaces
  - 1. Registration — `/auth?mode=up` (`src/routes/auth.tsx`)
  - 2. Login — `/auth?mode=in`
  - 3. Verification — `/verify` (`src/routes/verify.tsx`)
  - 4. Home — `/` (`src/routes/index.tsx`)
  - 5. Packages / Subscription — `/plans` (`src/routes/plans.tsx`)
  - 6. Checkout / Payment — `/checkout/$id` (`src/routes/checkout.$id.tsx`)
  - 7. Health — `/health` (`src/routes/health.tsx`)
  - 8. Test yourself — `/test` (`src/routes/test.tsx`)
  - 9. Sport and food — `/exercises` (`src/routes/exercises.tsx`)
  - 10. My dashboard — `/dashboard` (`src/routes/dashboard.tsx`)
  - 11. Admin panel — `/admin` (`src/routes/admin.tsx`)
  - 12. Community — `/community` (`src/routes/community.tsx`)
  - 13. Post detail — `/posts/$id`
  - 14. New post — `/new`
  - 15. Profile — `/profile`
  - 16. Documentation — `/about`
- 05 — User Workflows
  - 1. Registration and login
  - 2. Subscriber: group registration (capacity)
  - 3. Attendance
  - 4. Specialist content submission
  - 5. Admin review, approval and rejection
  - 6. Resubmission
  - 7. Questions and alerts
  - 8. User management
  - 9. Community participation
- 06 — Use Case Model

    - Actors
    - Use case diagram
    - Use case list
    - Detailed descriptions
- 07 — Object-Oriented Analysis and Design
    - Approach
    - Classes
    - Relationships
    - Class diagram
    - Design decisions (recorded in `AGENTS.md`)
- 08 — Database Design
    - Overview
    - ERD
    - Entities
    - Enum types
    - Business-rule triggers and functions
    - Seed data
    - Mapping to the presentation's ERD
- 09 — UML Diagrams
    - Use case
    - Class
    - Activity diagrams
    - Sequence diagrams
    - State diagrams
    - Component diagram
    - Deployment diagram
- Appendix A — Database Schema (generated from the live database)
    - `alerts`
    - `attendance`
    - `comments`
    - `devices`
    - `member_contacts`
    - `payment_methods`
    - `payments`
    - `post_likes`
    - `posts`
    - `profiles`
    - `queries`
    - `subscription_groups`
    - `subscriptions`
    - `user_roles`
    - Enumerated types
    - Indexes
    - Database functions (full bodies)
    - Migration history
- 10 — System Architecture
    - Frontend
    - Application structure
    - Database
    - Authentication
    - Data flow
    - Server-side code
    - External services
- 11 — Source Code Documentation
    - Project file structure
    - Key modules
    - Database logic (real SQL)

    - Validation logic summary
    - Role-based access control in code
    - State management
- 12 — UI / UX Documentation
    - Main layout
    - Navigation
    - RTL / Arabic
    - Visual design
    - Forms
    - Buttons, cards, tables
    - Dashboards
    - Responsive behaviour
    - Validation and error states
    - Screenshots
- 13 — Security
    - Authentication
    - Authorization and RBAC
    - Row-level security (54 policies, all tables)
    - Validation
    - Access restrictions in the UI
    - Security scan status
    - Not implemented / not claimed
- 14 — Testing
    - Test approach and evidence
    - Defects found
- 15 — Traceability Matrix
- 16 — Limitations and Future Enhancements
    - Limitations (current state, 27 Sep 2026)
    - Future Enhancements — NOT CURRENTLY IMPLEMENTED
- 17 — Academic Summary
- Documentation Audit
    - What was documented
    - Diagrams created (`diagrams/`, each as .mmd + .svg + .png)
    - Screenshots (`screenshots/`)
    - Source files analysed
    - Database entities documented
    - Workflows documented
    - Could not be verified
    - Issue discovered during the audit
    - Labels used

My Fitness Community (لياقتي) — Documentation Package

Academic and technical documentation for the **My Fitness** digital fitness community platform, a Management Information Systems (MIS) academic project.

All content is derived from the existing source code in this repository, the live project database catalog (tables, access rules, triggers, functions), and the original project presentation that the project was built from. Anything not verifiable is labelled **Inferred**, **Not implemented**, or **Future Enhancement — NOT CURRENTLY IMPLEMENTED**.

- Live site: https://fit-fam-circle.lovable.app
- Documentation date: 27 September 2026

Index

#	Document	Contents
01	Project Overview	Purpose, problem, solution, objectives, scope, technologies
02	Requirements Analysis	Functional / non-functional requirements, implementation status
03	System Roles	Subscriber, Specialist, Admin: permissions and restrictions
04	System Interfaces	Every page: inputs, outputs, validation, entities, screenshots
05	User Workflows	Step-by-step implemented workflows
06	Use Cases	Actors, use case diagram, detailed use case descriptions
07	Object-Oriented Analysis	Classes, attributes, responsibilities, class diagram
08	Database Design	Entities, keys, relationships, constraints, ERD
08b	Appendix A — Database Schema	Generated column list, policies, triggers, function bodies
09	UML Diagrams	All UML diagrams and applicability
10	System Architecture	Frontend, backend, auth, data flow
11	Source Code Documentation	File tree, key files, functions, access control code
11b	Appendix B — Source Code	Complete application source
12	UI / UX Documentation	Layout, RTL, forms, states, responsiveness
13	Security	Implemented security mechanisms only
14	Testing	Test cases with observed results
15	Traceability Matrix	Requirement → interface → use case → entity → code → test
16	Limitations and Future Enhancements	Honest limitations, labelled future work
17	Academic Summary	Problem, analysis, design, results, author's role
—	Documentation Audit	What was verified and what could not be

Folders

- `diagrams/` — Mermaid source (`.mmd`) plus rendered `.svg` and `.png` for every diagram.
- `screenshots/` — Screenshots of the running site captured on 27 Sep 2026.
- `My-Fitness-Documentation.html` / `.pdf` — single-file combined versions of documents 01–17.

Conventions

- UI text is Arabic (RTL). Arabic labels are quoted with an English translation.
- Database entity names are written in `code` style and match the real table names.
- "Specialist" and "Trainer" (مدرب متخصص) mean the same role (`specialist` in the database).

01 — Project Overview

Project title

My Fitness (لياقتي) — Digital Fitness Community Platform

Academic context

Management Information Systems (MIS) academic project. The platform is a working prototype that implements the analysis and design documented in the original project presentation (use cases, activity diagrams, ERD and eight documented interfaces). It is **not** a production medical, fitness-coaching or payment service. The site footer states this: "المحتوى تثقيفي عام وليس استشارة طبية أو تدريبية متخصصة" ("Content is general education, not medical or specialised training advice").

Purpose

Connect people interested in fitness with specialist trainers in one Arabic-language platform where members can learn, share challenges, solutions and experiences, join supervised subscription groups, track attendance and ask trainers questions, while an administrator keeps content and membership under control.

Problem statement

Fitness information online is scattered and often inaccurate, and there is no single place where trainees can (a) find reviewed guidance about exercises, equipment and sports nutrition, (b) join a supervised group with a trainer, and (c) interact with a community, with an administrator checking content before publication.

Proposed solution

A web application with three roles:

- **Subscriber (trainee)** — registers, explores content, subscribes to a group, records attendance, asks questions, publishes community posts.
- **Specialist (trainer)** — writes exercise/device and nutrition guidance, answers questions, reviews attendance, sends alerts.
- **Admin** — manages users, trainer role, groups, capacity, payment methods, subscription requests, attendance, and approves or rejects content.

Objectives (from the presentation, reflected on the Home page)

1. Provide correct information about sports exercises.
2. Provide information about sports nutrition.
3. Spread sports culture.
4. Provide tools for tracking health and fitness (BMI and water calculator, self-tests, attendance log).
5. Provide educational and cultural content.

Target users

Arabic-speaking trainees, fitness trainers, and a platform administrator.

Scope

Functional scope (implemented)

Registration with email confirmation, login, role-based dashboards, subscription groups with capacity, checkout with offline payment methods (recorded only), admin approval of subscriptions, attendance recording and review, questions and answers,

trainer alerts, trainer guidance with admin approval, community posts with comments and likes and admin approval, health article + calculators, self-assessment quizzes, profile management, project documentation page.

Non-functional scope (implemented)

Arabic RTL interface, responsive layout (desktop and mobile navigation), database-level access control (row-level security), server-side business-rule enforcement via database triggers, SEO metadata per page.

Out of scope

Real online payment processing, medical diagnosis, mobile native apps, online shop, real-time chat, social-login providers (see document 16).

Technologies actually used

Verified from `package.json` and source code:

Layer	Technology
UI framework	React 19
Full-stack framework / routing	TanStack Start v1, TanStack Router (file-based routes in `src/routes/`)
Build tool	Vite 7
Data fetching / cache	TanStack Query v5
Styling	Tailwind CSS v4 (`src/styles.css`), shadcn/ui components on Radix UI
Icons	lucide-react
Validation	Zod
Notifications	sonner (toasts)
Backend	Lovable Cloud (managed Supabase): Postgres database, Auth, Data API
Client SDK	`@supabase/supabase-js`
Fonts	Cairo (headings), Tajawal (body) — loaded via `<link>` in `__root.tsx`
Hosting	Lovable hosting (edge runtime), published at fit-fam-circle.lovable.app

No custom server functions or external APIs are used; the browser talks directly to the backend, and all protection is enforced in the database (see document 13).

02 — Requirements Analysis

Status legend: **Implemented** (verified in code/database) · **Partial** · **Academic only** (in the presentation's analysis but not built) · **Not implemented**.

Functional requirements

ID	Requirement	Actor	Status	Evidence
FR-01	Register with name, email, password (min 6), optional phone and age	Visitor	Implemented	`src/routes/auth.tsx` (`signupSchema`), trigger `handle_new_user`
FR-02	Verify email before sign-in; resend confirmation link	Subscriber	Implemented	`src/routes/verify.tsx` (`supabase.auth.resend`)
FR-03	Log in with email and password; redirect to dashboard	All	Implemented	`auth.tsx` (`signInWithPassword`)
FR-04	Log out	All	Implemented	`SiteHeader.tsx` (`signOut`)
FR-05	Every new user gets `subscriber` role; the first user ever also gets `admin`	System	Implemented	`handle_new_user()`
FR-06	Admin grants/removes the specialist role	Admin	Implemented	`admin.tsx` → `Users`
FR-07	Admin blocks/unblocks a member ("delete" in the presentation = block)	Admin	Implemented	`profiles.status`, `protect_profile_status()`
FR-08	Blocked members cannot post, comment, subscribe, ask or record attendance	System	Implemented	RLS `is_active_member()` checks
FR-09	Member edits own profile (name, bio, brief, level, phone, age)	All	Implemented	`profile.tsx`, `member_contacts`
FR-10	View subscription groups (packages) with price, features, seats	Visitor	Implemented, **see defect D-01**	`plans.tsx`
FR-11	Check group space before registration; show "لا يوجد مكان في المجموعة" when full	Subscriber	Implemented	`isFull()`, `check_group_space()`
FR-12	Prevent registering twice in the same group	System	Implemented	`check_group_space()` → `ALREADY_REGISTERED`
FR-13	Update group subscriber count on registration / rejection / deletion (deletion is database-only; no UI button)	System	Implemented	`after_subscription_change()`, `sync_group_count()`
FR-14	Choose payment method and submit subscription request (no real payment)	Subscriber	Implemented	`checkout.$id.tsx`, `payments`
FR-15	Admin accepts/rejects subscription requests (payments confirmed/rejected)	Admin	Implemented	`admin.tsx` → `Requests`
FR-16	Admin creates groups, toggles active, sets/updates capacity (not below current count)	Admin	Implemented	`admin.tsx` → `Groups`
FR-17	Admin manages payment methods (add, activate/deactivate)	Admin	Implemented	`admin.tsx` → `Methods`
FR-18	Subscriber records attendance (today only, active group, once per day, within group dates)	Subscriber	Implemented	`dashboard.tsx` → `MyAttendance`, `validate_attendance()`
FR-19	Subscriber deletes own attendance before review	Subscriber	Implemented	RLS "Delete own unreviewed attendance"
FR-20	Specialist reviews attendance (approve / reject + note)	Specialist	Implemented	`dashboard.tsx` → `ReviewAttendance`
FR-21	Admin views, filters by group, changes status, deletes attendance	Admin	Implemented	`admin.tsx` → `AttendanceAdmin`

ID	Requirement	Actor	Status	Evidence
FR-22	Subscriber asks a question	Subscriber	Implemented	`MyQueries`, `queries`
FR-23	Specialist answers questions	Specialist	Implemented	`AnswerQueries`
FR-24	Specialist sends alerts to subscribers; subscriber marks them read	Specialist / Subscriber	Implemented	`alerts`, `MyAlerts`
FR-25	Specialist writes device/exercise/nutrition/culture guidance (instructions, advantages, disadvantages, optional group)	Specialist	Implemented	`exercises.tsx` → `GuideForm`, `devices`
FR-26	Specialist edits/deletes own guidance; sees "Content I manage"	Specialist	Implemented	`exercises.tsx`, `dashboard.tsx` → `MyContent`
FR-27	Content submission: non-admin guidance and posts start as **pending**	System	Implemented	`enforce_content_approval()`
FR-28	Content review: admin lists pending guidance and posts	Admin	Implemented	`admin.tsx` → `ContentReview`
FR-29	Approval: admin approves → content becomes public	Admin	Implemented	RLS "Admins review posts/guidance"
FR-30	Rejection: admin must enter a reason; author and admin see status + reason; hidden from others	Admin	Implemented	`rejection_reason`, RLS
FR-31	Resubmission: author edit of approved/rejected content returns it to pending and clears reason	Specialist / Subscriber	Implemented	`enforce_content_approval()`
FR-32	Browse community posts, filter by type/category, search by title	Visitor	Implemented	`community.tsx`, `fetchPosts()`
FR-33	Publish post (challenge/solution/experience/tip + category)	Subscriber	Implemented	`new.tsx`
FR-34	Comment, like/unlike, delete own post/comment	Member	Implemented	`posts.$id.tsx`
FR-35	Health page: article + BMI and daily water calculator	Visitor	Implemented	`health.tsx`
FR-36	Self-test page with four yes/no quizzes	Visitor	Implemented	`test.tsx`
FR-37	Documentation page (requirements, UML, ERD in Arabic)	Visitor	Implemented	`about.tsx`
FR-38	Session timetable for groups	—	Not implemented (not described in presentation)	—
FR-39	Subscriber self-cancellation of a subscription	—	Not implemented (not described in presentation)	—
FR-40	Real electronic payment	—	Not implemented (academic prototype)	—

Non-functional requirements (supported)

ID	Requirement	Status	Evidence
NFR-01	Arabic language, right-to-left layout	Implemented	`<html lang="ar" dir="rtl">` in `__root.tsx`
NFR-02	Responsive layout (mobile nav bar under header at < lg)	Implemented	`SiteHeader.tsx`, Tailwind breakpoints; mobile screenshots
NFR-03	Access control enforced by the database, not only the UI	Implemented	54 RLS policies (document 13)

ID	Requirement	Status	Evidence
NFR-04	Business rules enforced even if the UI is bypassed	Implemented	triggers `check_group_space`, `validate_attendance`, `enforce_content_approval`
NFR-05	Input length limits	Implemented	CHECK constraints + client `maxLength`/`slice`
NFR-06	Per-page title/description metadata	Implemented	`head()` in each route
NFR-07	Friendly error pages (404, error boundary)	Implemented	`NotFoundComponent`, `ErrorComponent` in `__root.tsx`
NFR-08	Performance/load targets	Not specified / not measured	—
NFR-09	Accessibility conformance level	Not verified	Some `role="alert"` usage only

Requirements represented in the academic project but not implemented

- Online shop / cart, social media links, dark mode: not in the presentation's use cases/ERD, deliberately not built.
- A measured availability or performance SLA.
- Real payment gateway (presentation shows payment interface; implemented as a request record only).

03 — System Roles

Roles are stored in the `user_roles` table (enum `app_role`: `admin`, `specialist`, `subscriber`) and checked in the database with `has_role(user_id, role)`. A user can hold several roles (e.g. the first user is `subscriber` + `admin`). The browser loads roles in `AuthProvider` (`src/lib/auth.tsx`) only to decide which UI to show; every write is re-checked by the database.

Visitor (not signed in)

- **Can:** view Home, Sport and food (approved guidance), Health, Test, Packages list, Community (approved posts), post detail, Documentation, Register, Login, Verify.
- **Cannot:** subscribe, post, comment, like, ask, see profiles (author names show as "عضو" / Member).

Subscriber (مشترك)

Aspect	Detail
Responsibilities	Register, explore content, subscribe to a group, record attendance, ask questions, publish content
Interfaces	All visitor pages + Checkout, My dashboard (Subscriptions, Attendance, Questions, Alerts), New post, Profile
Allowed actions	Insert own subscription (pending) and payment (pending); insert own attendance for today; delete own unreviewed attendance; insert/delete own questions; mark own alerts read; insert posts (pending), comments, likes; edit own posts (returns to pending); edit own profile and contact
Restrictions	Cannot change subscription/payment status; cannot review attendance; cannot see other members' profiles, contacts, payments; cannot publish without admin approval; blocked members cannot write
Auth behaviour	Must confirm email before first login; session kept by the auth client; header shows "لوحتي", "ملفي", "خروج"

Specialist / Trainer (مدرب متخصص)

Aspect	Detail
Responsibilities	Write exercise/device, nutrition and culture guidance; answer questions; review attendance; send alerts; manage own content
Interfaces	Everything a subscriber has + Specialist tabs in My dashboard (Content I manage, Questions, Attendance review, Alerts) + guidance form on Sport and food page
Allowed actions	Insert guidance (`devices`) as self → pending; edit own guidance (→ pending); delete own guidance; update queries to answer them (as self); update attendance status/note; insert alerts as self; read all subscriptions, questions and attendance
Restrictions	Cannot approve content; cannot manage users, groups, payments; cannot edit other trainers' guidance
How assigned	Admin toggles "set as trainer" in Admin → Users

Admin (مشرف)

Aspect	Detail
Responsibilities	Manage users and trainer role, subscription/participation requests, groups and capacity, payment methods, attendance, content approval
Interfaces	Admin page tabs: Users, Requests, Groups, Payment methods, Attendance, Content review; plus all other pages
Allowed actions	Manage `user_roles`; update any profile `status`; read contacts; update/delete subscriptions and payments; full CRUD on groups and payment methods; update/delete attendance; approve/reject posts and guidance (reason required for reject); delete any post, comment or guidance; admin-authored content auto-approved
Restrictions	None enforced beyond the app's feature set. Admin link appears only when `isAdmin`
How assigned	Automatically the first registered account (`handle_new_user`)

Authentication / authorization summary

- Authentication: email + password via Lovable Cloud Auth with email confirmation (`emailRedirectTo: window.location.origin`).
- Authorization: row-level security policies on all 14 public tables, using `auth.uid()` and `has_role()`; status protection trigger prevents members changing their own `status`.
- Client gating: pages such as Dashboard/Admin/New/Profile show "يرجى تسجيل الدخول" (Please sign in) when no session; Admin page additionally requires `isAdmin`. Client gating is convenience only; the database is the enforcement point.

04 — System Interfaces

The eight interfaces documented in the presentation (Registration, Login, Verification, Home, Subscription, Payment, Health, Test) map to the pages below, plus the additional implemented pages needed for the roles' workflows. Screenshots were captured on 27 Sep 2026 at 1280 px width (and 390 px for mobile samples).

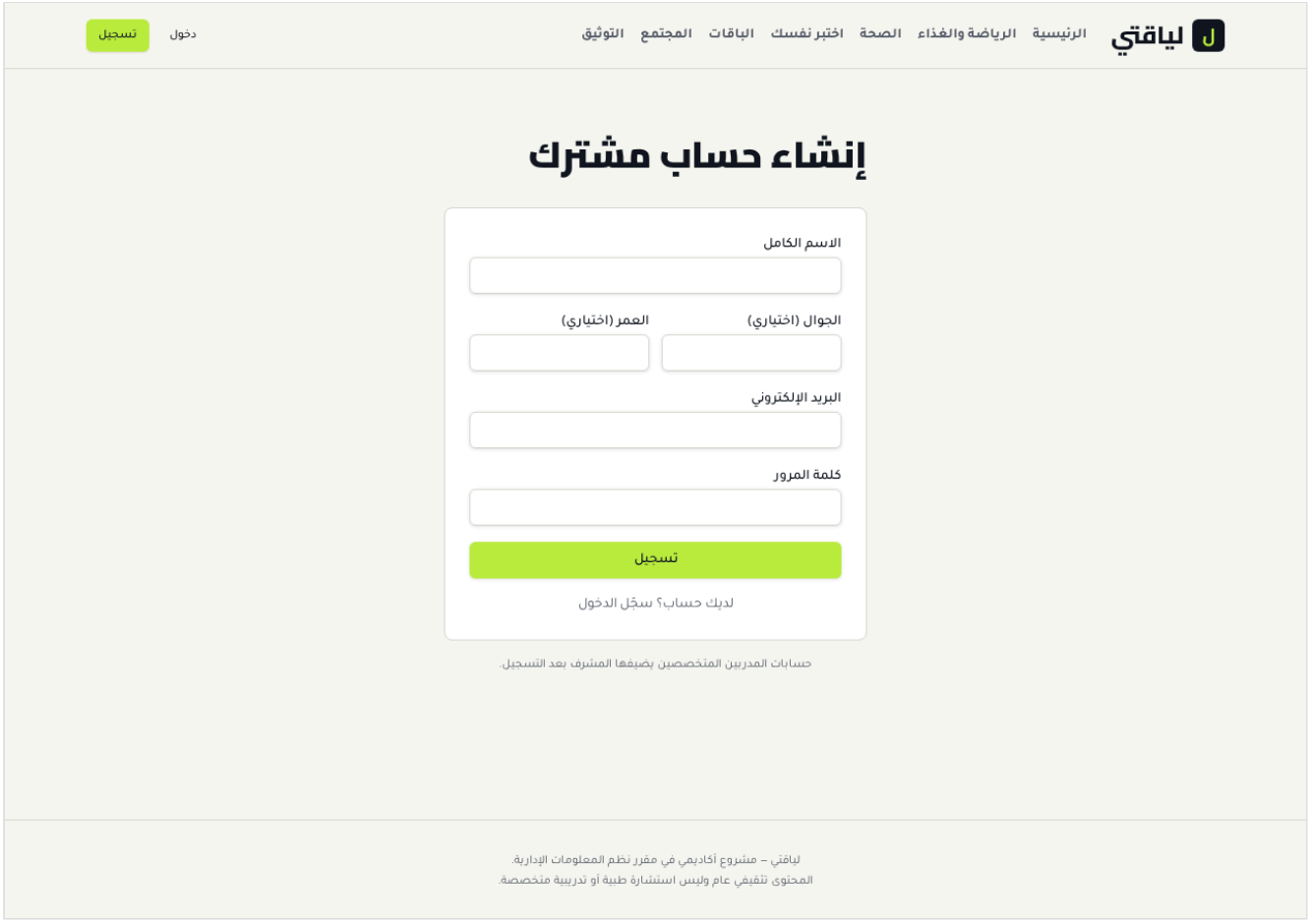
> **Screenshot limitation:** the project database currently has **no user accounts**, so signed-in screens (Dashboard, Admin, Profile, New post, Checkout) could not be captured in their signed-in state. Their signed-out state is shown where relevant, and their contents are documented from source code.

#	Route	Interface	Presentation interface
1	`/auth?mode=up`	Registration	1. Registration
2	`/auth?mode=in`	Login	2. Login
3	`/verify`	Verification	3. Verification
4	`/`	Home	4. Home
5	`/plans`	Packages / Subscription	5. Subscription
6	`/checkout/$id`	Checkout / Payment	6. Payment
7	`/health`	Health	7. Health
8	`/test`	Test yourself	8. Test
9	`/exercises`	Sport and food (guidance)	Supports trainer "provide guidance" use case
10	`/dashboard`	My dashboard	Supports subscriber/trainer use cases
11	`/admin`	Admin panel	Supports admin use cases
12	`/community`	Community	Share challenges/solutions/experiences
13	`/posts/$id`	Post detail	Community
14	`/new`	New post	Publish content
15	`/profile`	My profile	Manage account
16	`/about`	Documentation	Academic documentation
—	any unknown URL	404 page	—

---

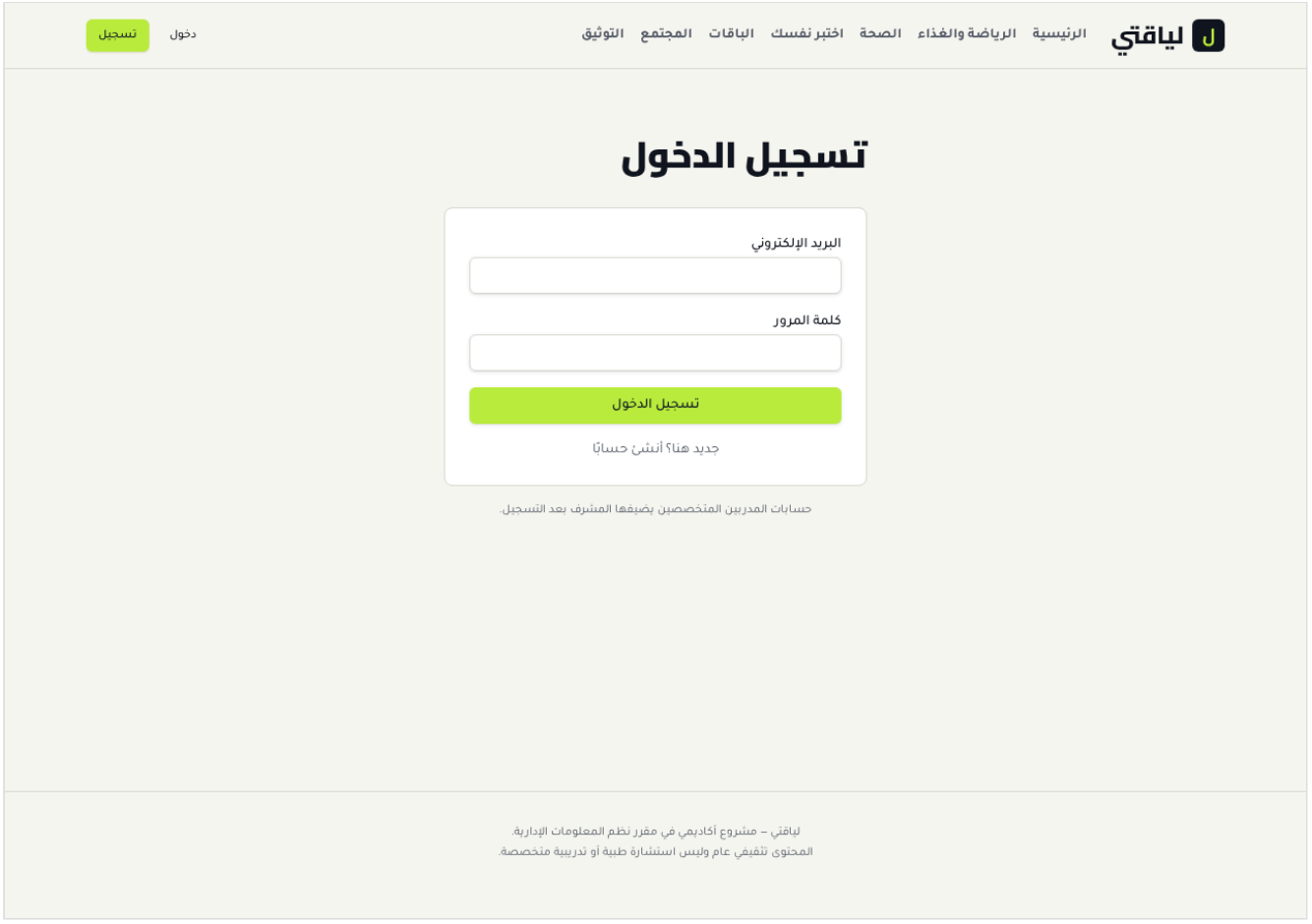
1. Registration — `/auth?mode=up` (`src/routes/auth.tsx`)

- **Purpose:** create a subscriber account. **Role:** Visitor.
- **Inputs:** name (2–50), email, password (≥ 6), phone (optional, digits/+/space ≤ 20), age (optional, 10–100).
- **Outputs:** confirmation email sent; redirect to `/verify?email=…`.
- **Validation:** Zod `signupSchema`; first error shown as a toast.
- **Workflow:** Authentication (05 §1). **Entities:** auth user, `profiles`, `member_contacts`, `user_roles` (created by trigger).
- **Navigation:** header "تسجيل"; links to login mode.



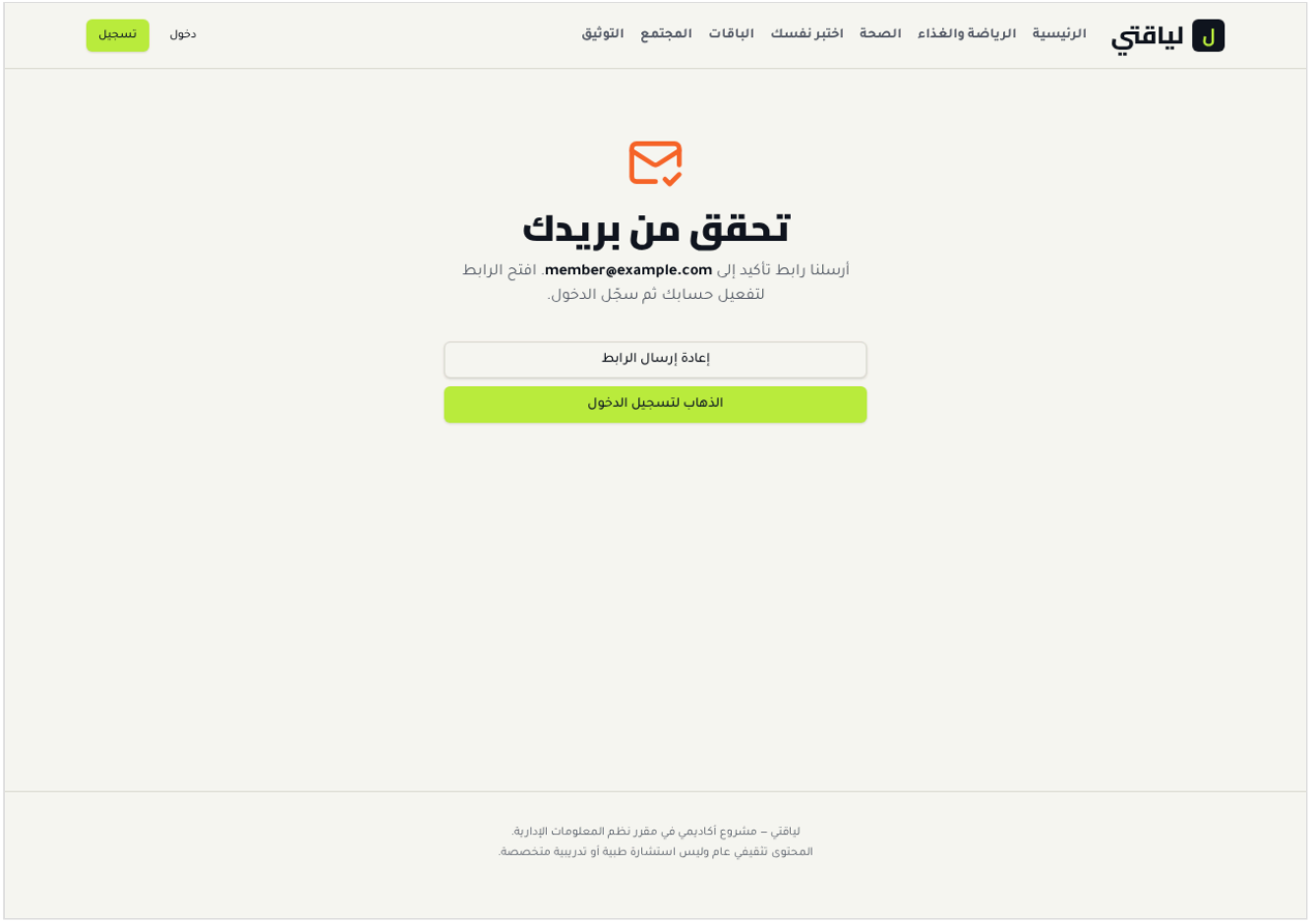
2. Login — `/auth?mode=in`

- **Inputs:** email, password. **Outputs:** session; redirect to `/dashboard`.
- **Validation:** `loginSchema`; unconfirmed email → `/verify`; wrong credentials → "البريد الإلكتروني أو كلمة المرور غير صحيحة".
- **Entities:** auth session, `user_roles`, `profiles.status` (loaded by `AuthProvider`).



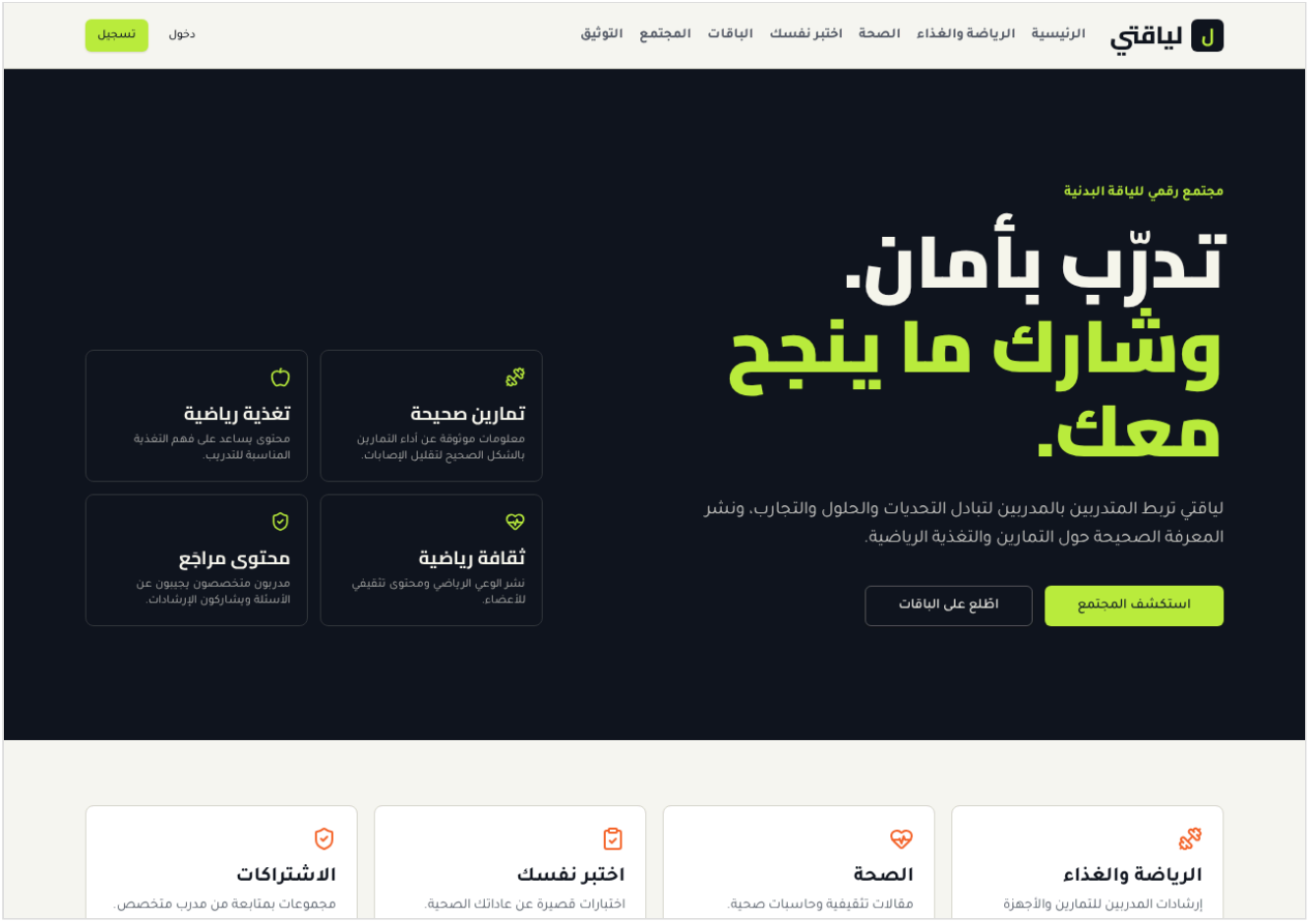
3. Verification — `/verify` (`src/routes/verify.tsx`)

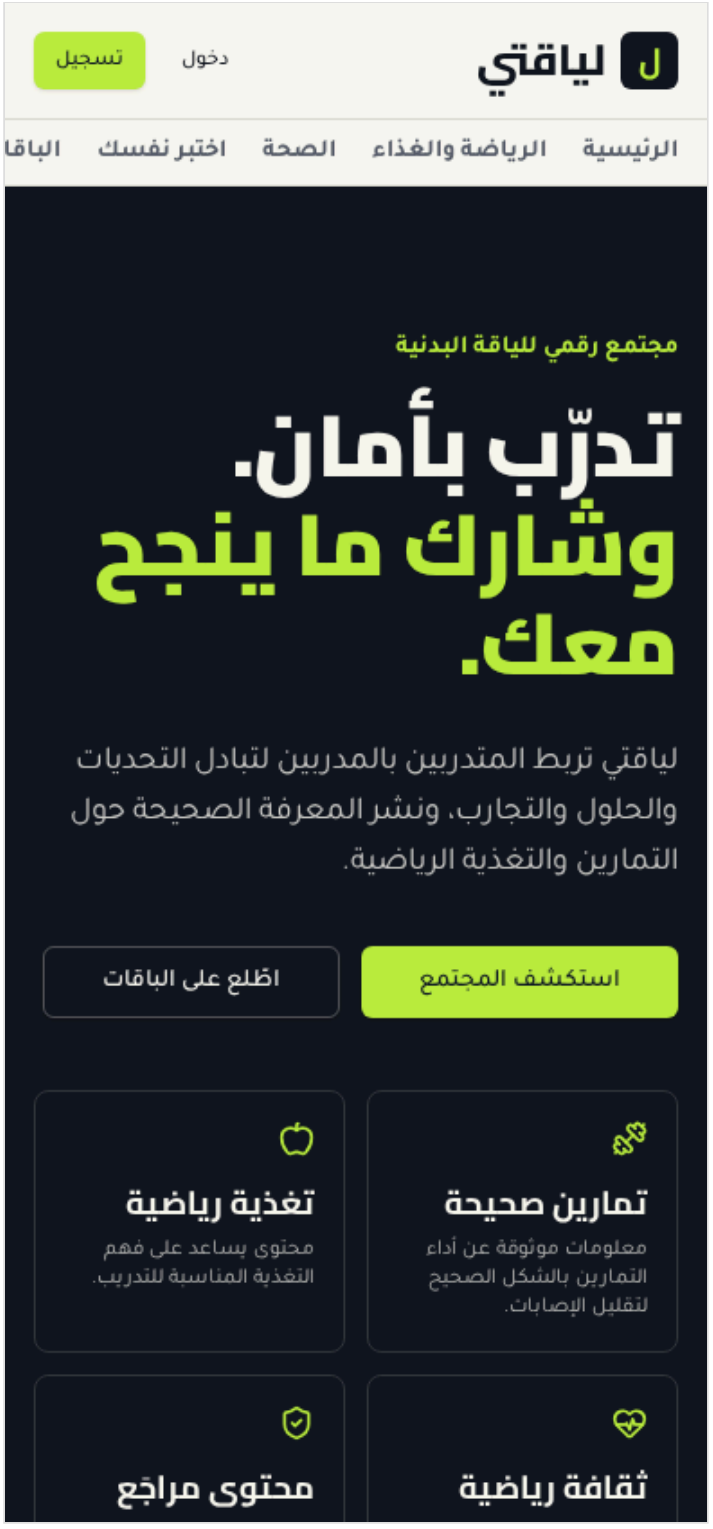
- **Purpose:** tell the user to confirm email; **action:** "إعادة إرسال الرابط" (resend) via `supabase.auth.resend`; link to login.



4. Home — `/` (`src/routes/index.tsx`)

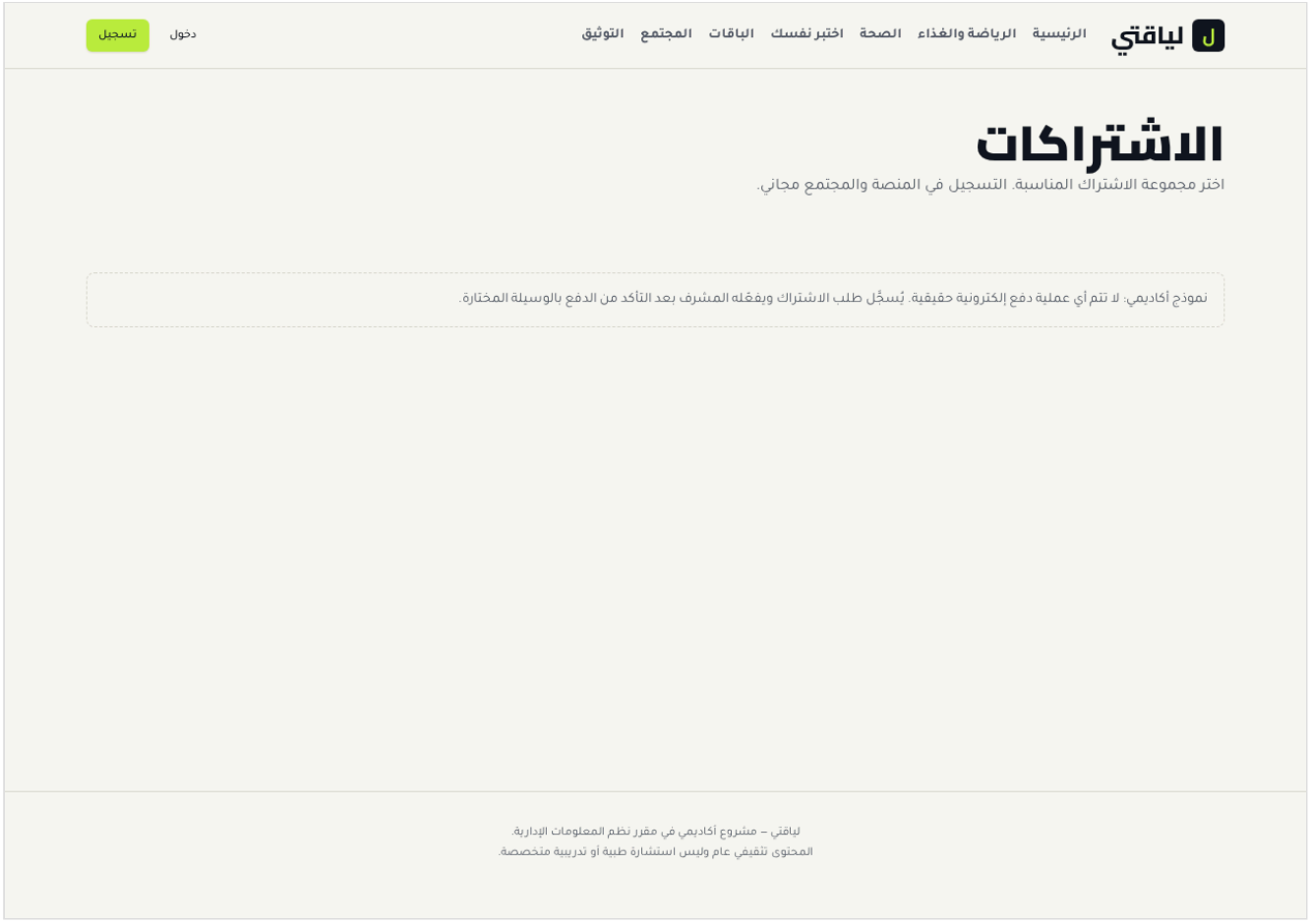
- **Purpose:** introduce the platform, its five objectives, the three roles, quick links to sections, and latest approved posts.
- **Outputs:** feature cards linking to `/exercises`, `/health`, `/test`, `/plans`; latest community posts (`fetchPosts({limit})`).
- **Entities:** `posts` (approved only).


لياقتي
الرئيسية
الرياضة والغذاء
الصحة
اختبر نفسك
الباقات
المجتمع
التوثيق
دخول
تسجيل
مجتمع رقمي للياقة البدنية
تدرّب بأمان.
وشارك ما ينجح معك.
لياقتي تربط المتدربين بالمدربين لتبادل التحديات والحلول والتجارب، ونشر المعرفة الصحيحة حول التمارين والتغذية الرياضية.
استكشف المجتمع
اطّلع على الباقات
تمارين صحيحة
معلومات موثوقة عن أداء التمارين بالشكل الصحيح لتقليل الإصابات.
تغذية رياضية
محتوى يساعد على فهم التغذية المناسبة للتدريب.
ثقافة رياضية
نشر الوعي الرياضي ومحتوى تثقيفي للأعضاء.
محتوى مراجَع
مدربون متخصصون يجيبون عن الأسئلة ويشاركون الإرشادات.
الرياضة والغذاء
إرشادات المدربين للتمارين والأجهزة
الصحة
مقالات تثقيفية وحاسبات صحية.
اختبر نفسك
اختبارات قصيرة عن عاداتك الصحية.
الاشتراكات
مجموعات بمتابعة من مدرب متخصص.



5. Packages / Subscription — `/plans` (`src/routes/plans.tsx`)

- **Purpose:** list active subscription groups (seeded: Silver 50 SAR, Platinum 60 SAR).
- **Outputs:** title, description, price, dates, features, seats "المقاعد: X / Y" when capacity set.
- **Actions:** "اشترك الآن" → `/checkout/$id` (signed in) or "سجّل للاشتراك" → registration. Full group → "لا يوجد مكان في المجموعة".
- **Entities:** `subscription_groups`.
- **Observed defect D-01:** at capture time the package list is empty for visitors because the database returns "permission denied for function has_role" to non-signed-in requests (see 14-Testing TC-05 and 16-Limitations).

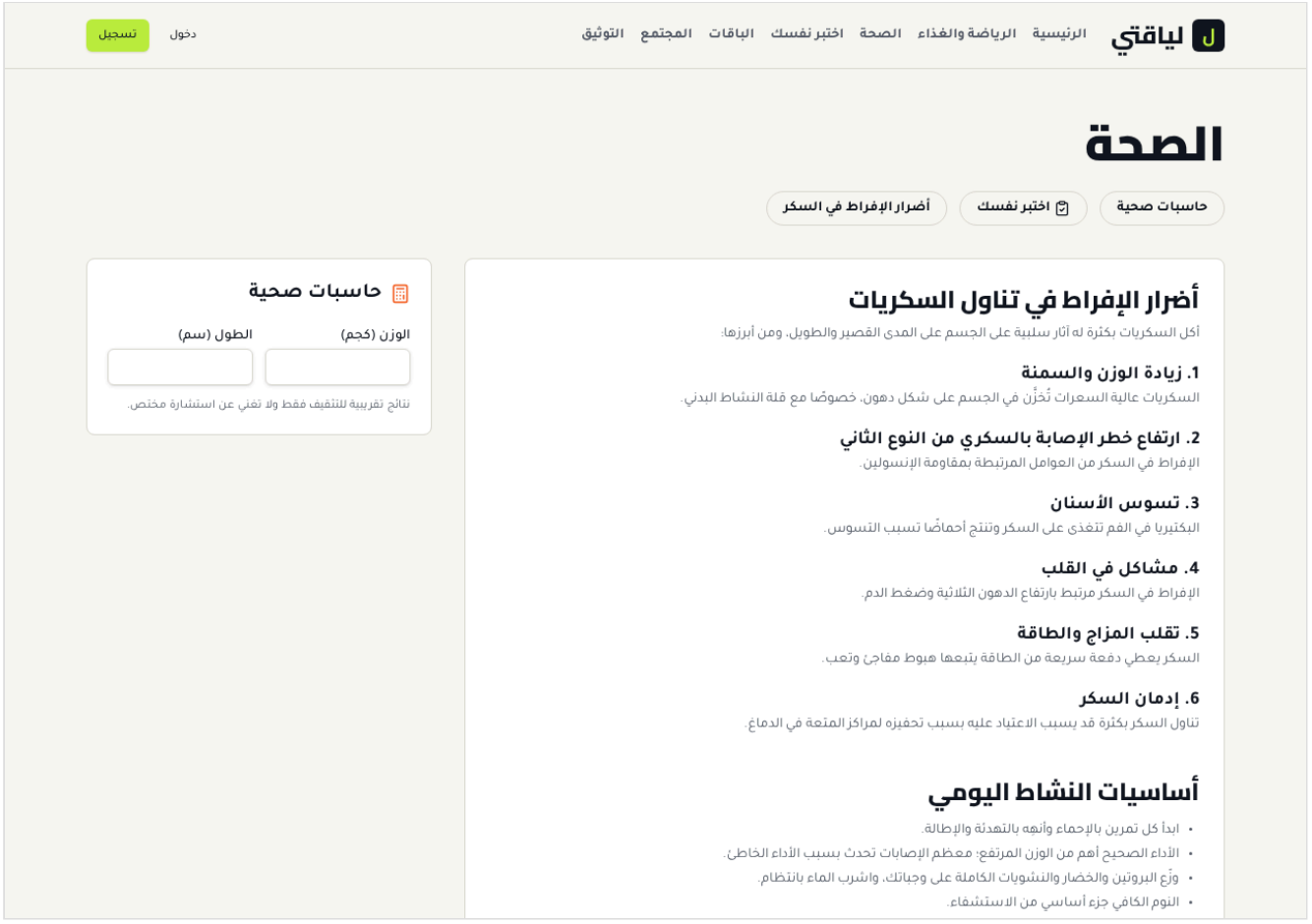


6. Checkout / Payment — `/checkout/$id` (`src/routes/checkout.$id.tsx`)

- **Role:** Subscriber (signed in, not blocked).
- **Inputs:** payment method (radio from active `payment_methods`), note (≤ 500).
- **Validation:** blocked check; method required; re-check duplicate subscription and fresh capacity; database trigger errors translated by `subscribeErrorMsg`.
- **Outputs:** `subscriptions` row (pending) + `payments` row (pending); toast "تم تقديم الطلب، بانتظار تأكيد المشرف".
- **Note:** no real money is transferred (stated on the page).
- Screenshot: not captured (requires a signed-in account).

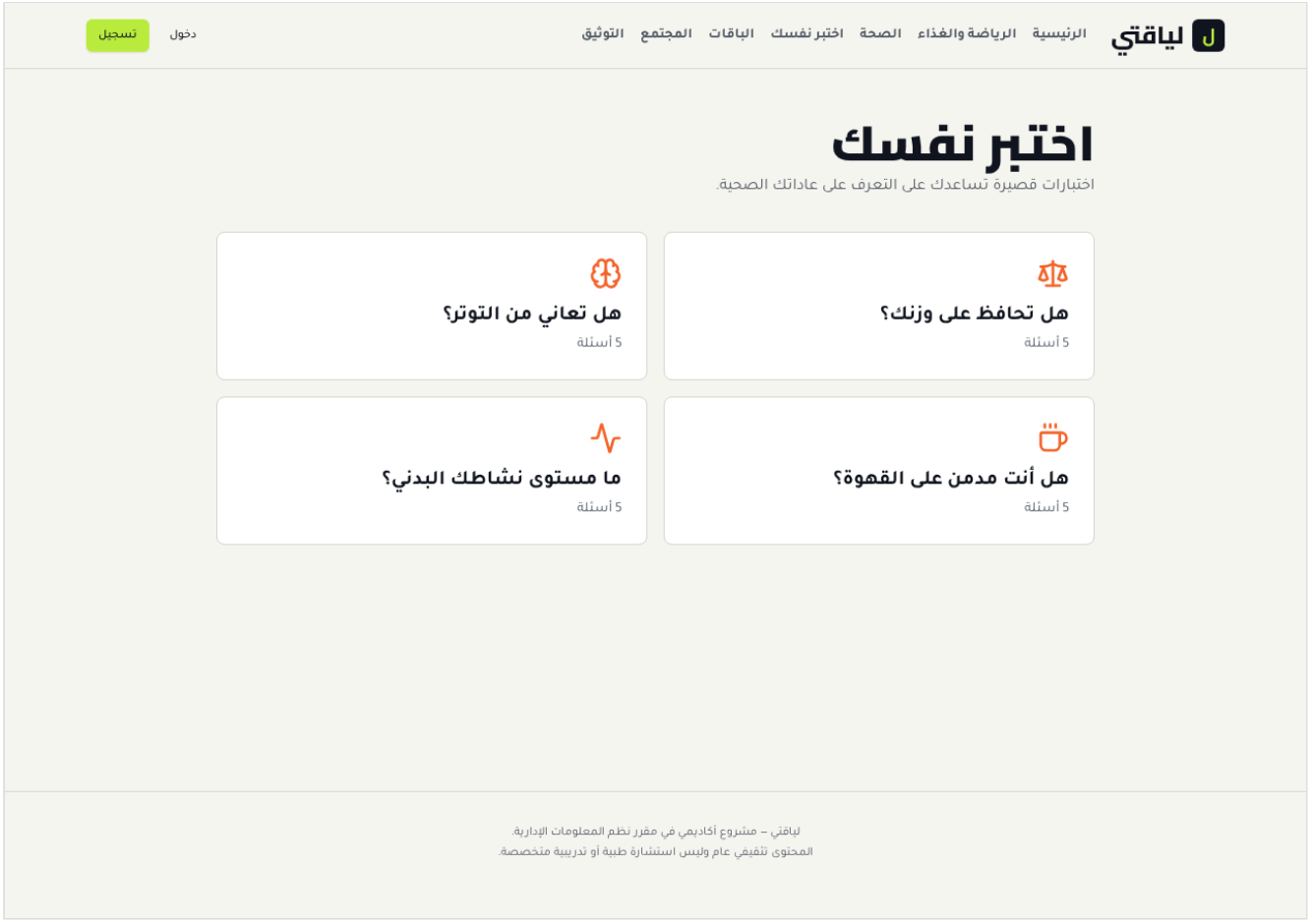
7. Health — `/health` (`src/routes/health.tsx`)

- **Content:** educational article on the harms of excess sugar; **BMI calculator** (weight, height → BMI and category label) and daily water estimate.
- No database access.



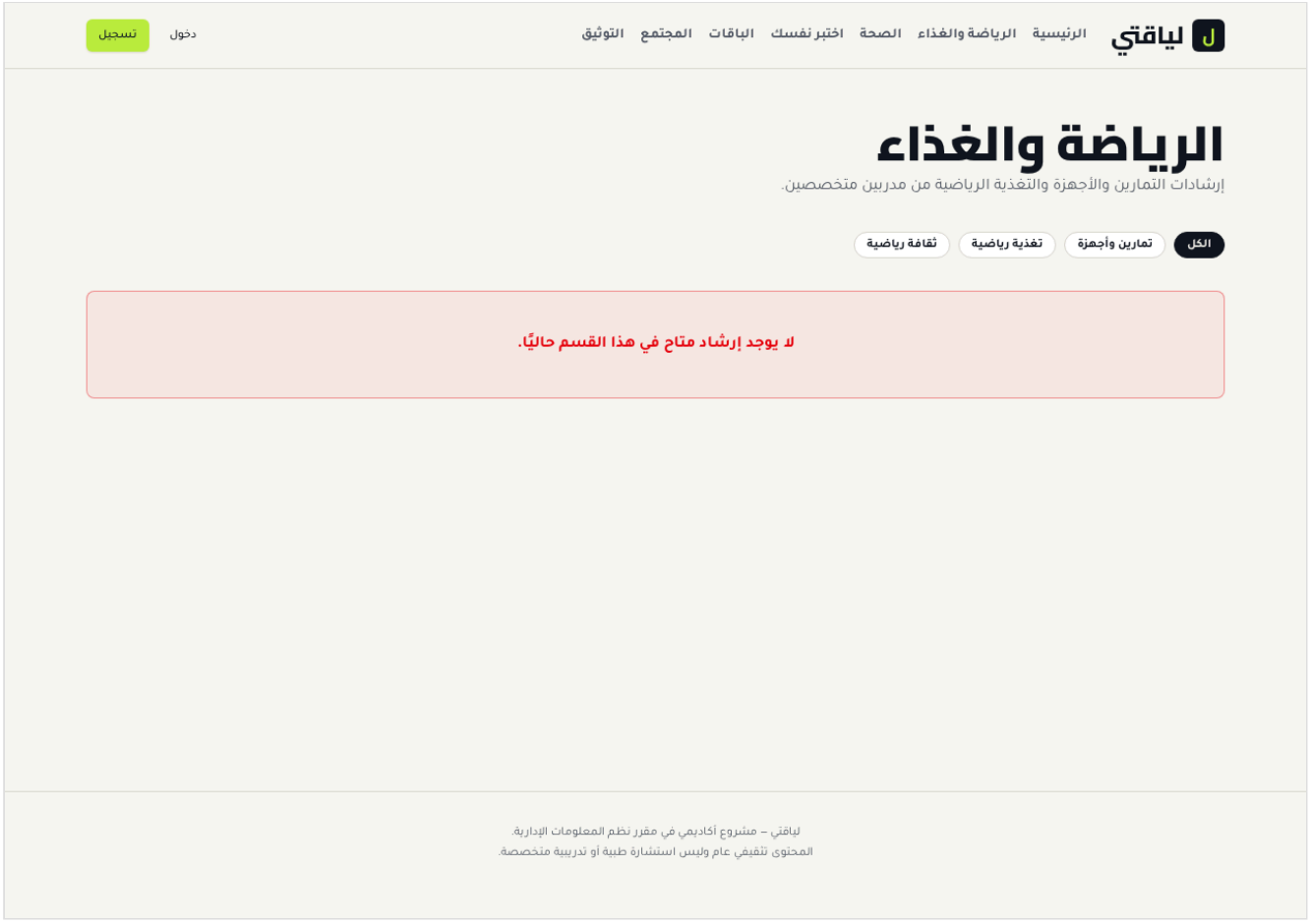
8. Test yourself — `/test` (`src/routes/test.tsx`)

- Four yes/no quizzes: weight habits, stress, coffee, physical activity. Score → result text. No data is stored.



9. Sport and food — `/exercises` (`src/routes/exercises.tsx`)

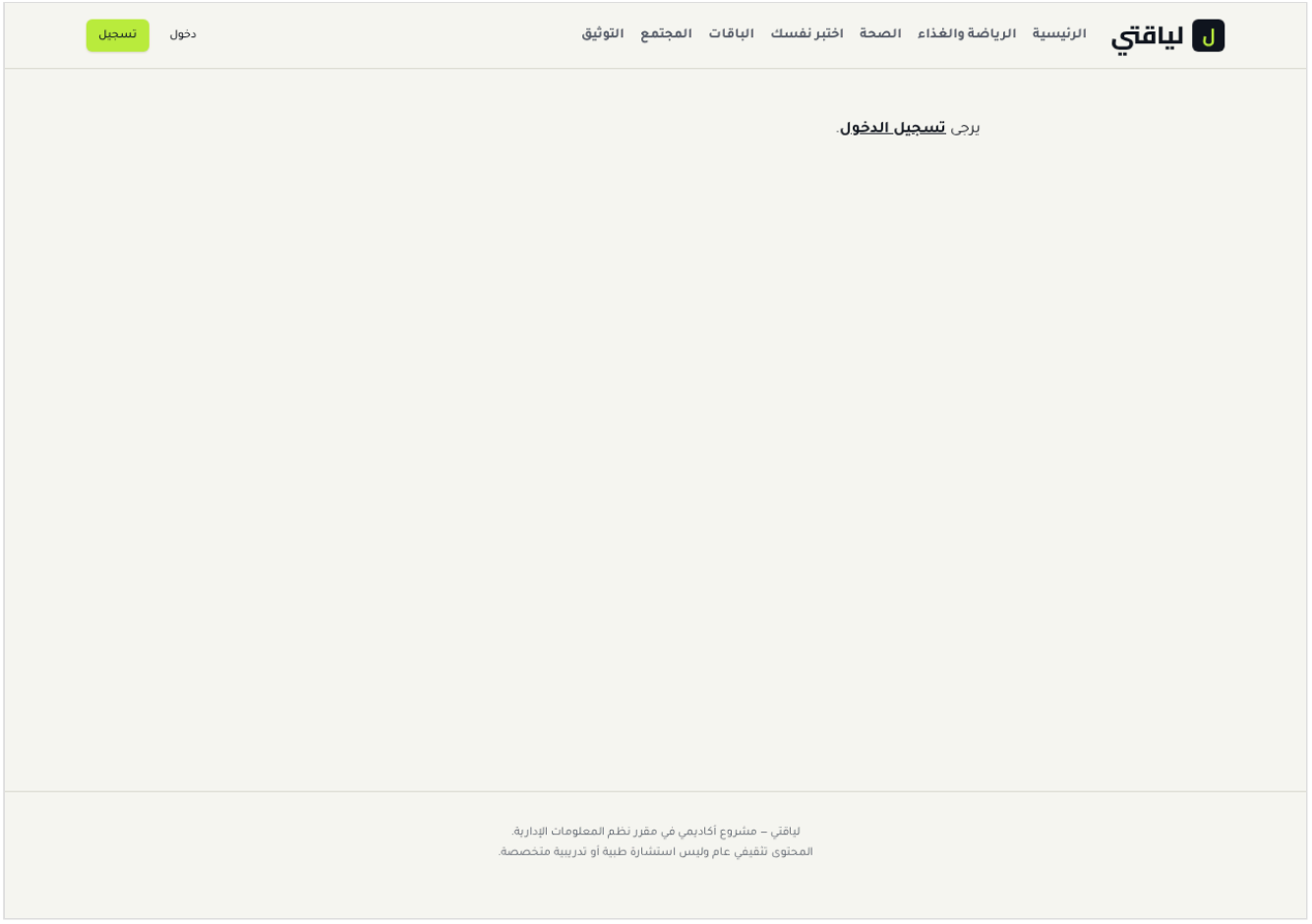
- **Visitors/subscribers:** approved guidance cards (device name, category: exercise/nutrition/culture, guidance, instructions, advantages, disadvantages, trainer, linked group), category filter, "اسأل المدرب عن هذا الإرشاد" link for subscribers. Empty state: "لا يوجد إرشاد متاح".
- **Specialist:** `GuideForm` to add/edit guidance (title ≥ 2, guidance ≥ 5), "إرشاداتي فقط" filter, pending/rejected badges and rejection reason on own items, delete own.
- **Admin:** can delete any guidance.
- **Entities:** `devices`, `subscription_groups`, `profiles`.



10. My dashboard — `/dashboard` (`src/routes/dashboard.tsx`)

Tabs: **المشترك** (Subscriber) for everyone and **المدرب** (Trainer) when `isSpecialist`.

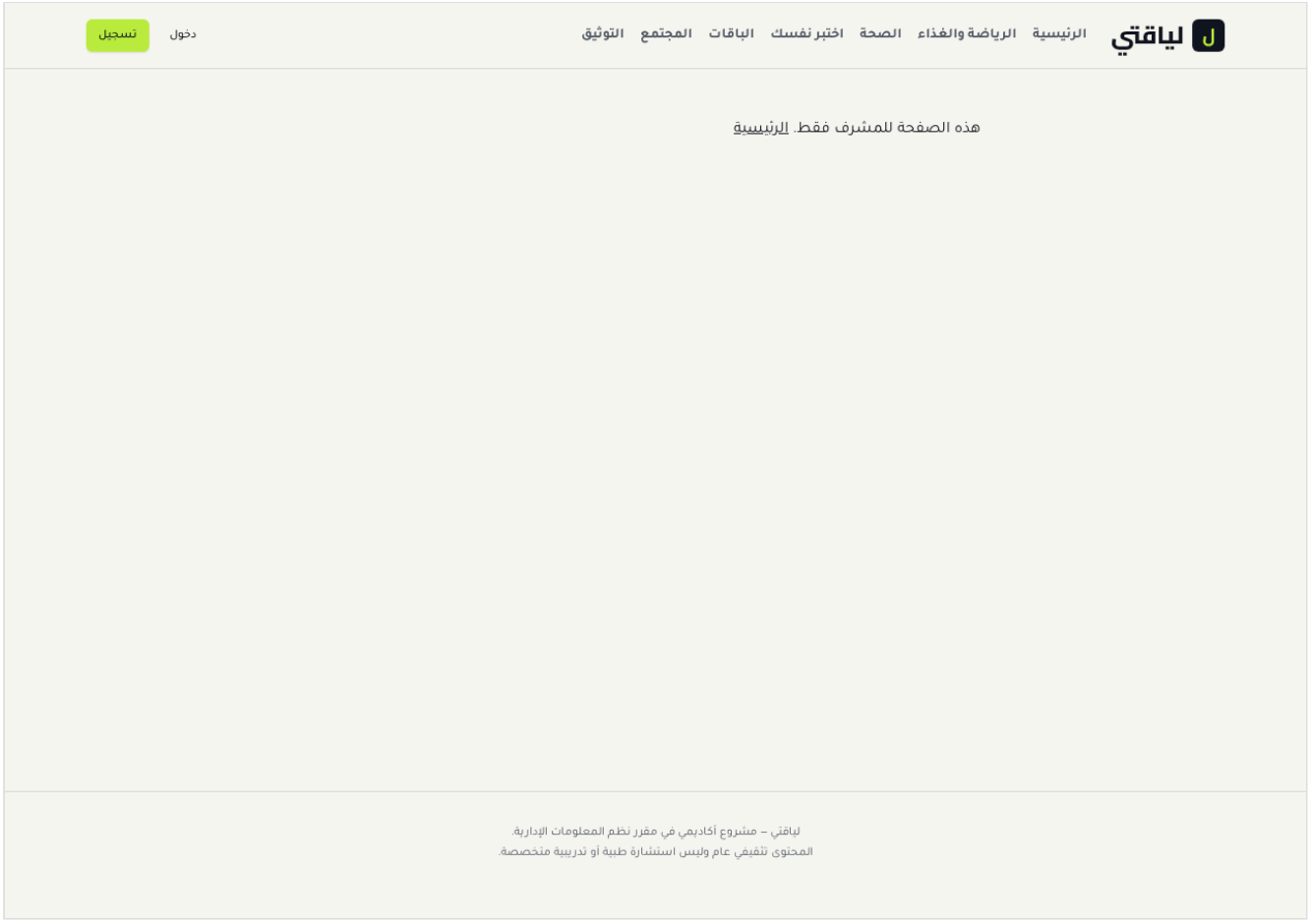
- Subscriber: `MySubscriptions` (status, group, payment), `MyAttendance` (group select, exercises, record, list with status/note, delete unreviewed; list of available groups with seats when not in a group), `MyQueries` (ask, see answers), `MyAlerts` (mark read).
- Trainer: `MyContent` (counts, assigned groups, own guidance/posts with status + reason), `AnswerQueries` (unanswered first, "عرض الكل"), `ReviewAttendance` (approve/reject + note), send alert.
- **Entities:** `subscriptions`, `payments`, `attendance`, `queries`, `alerts`, `devices`, `posts`, `subscription_groups`.
- Signed-out state:



11. Admin panel — `/admin` (`src/routes/admin.tsx`)

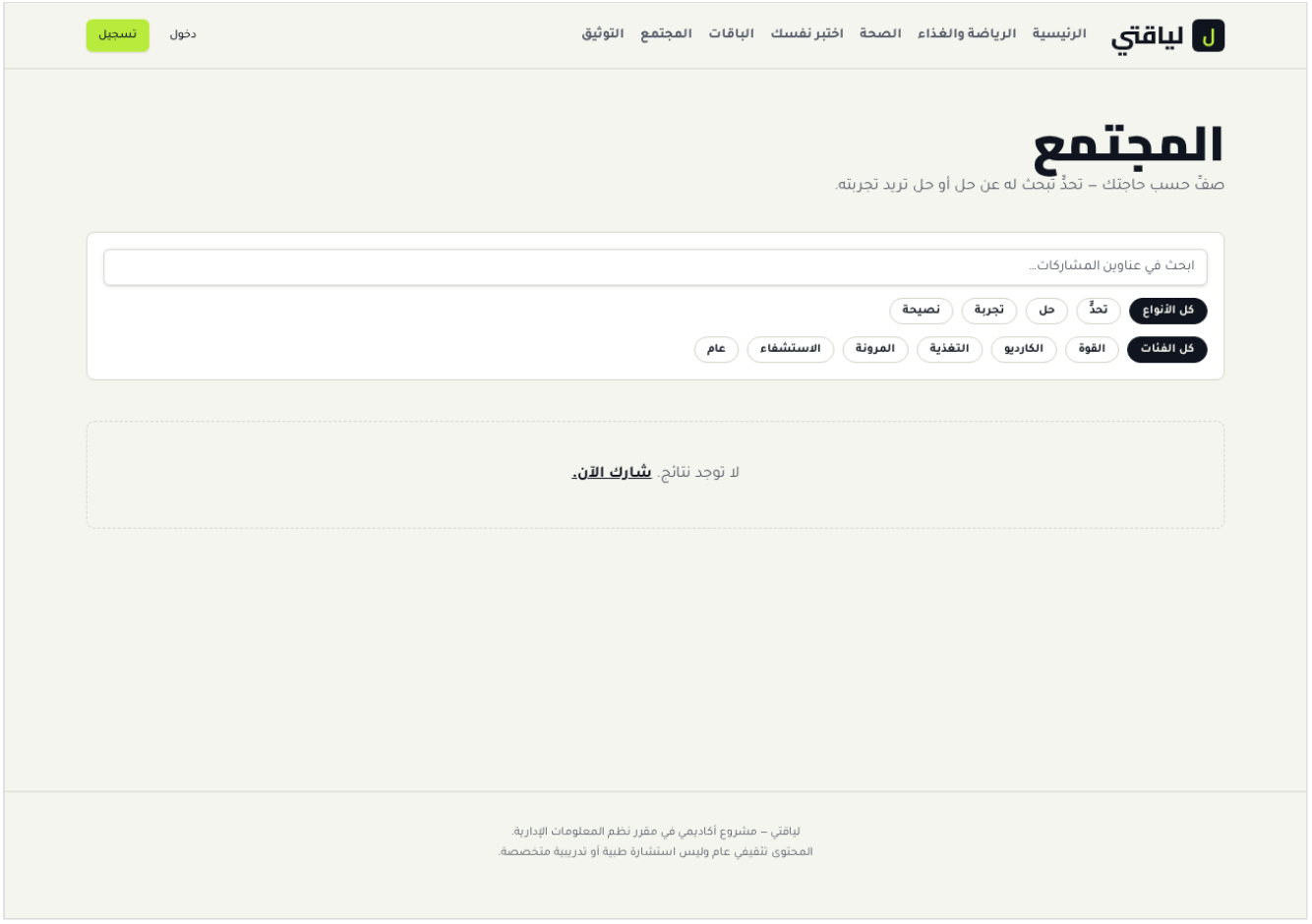
Only when `isAdmin`; otherwise "هذه الصفحة للمشرف فقط". Tabs:

- **المستخدمون** Users: name, phone, age, roles, status; toggle trainer; block/unblock.
- **الطلبات** Requests: subscription + payment details; accept / reject.
- **مراجعة المحتوى** Content review: pending guidance and posts; approve / reject (reason prompt, ≤ 500).
- **المجموعات** Groups: create (title, description, features, price, dates, capacity), activate/deactivate, update capacity (> 0, not below current count), "المشتركون: X / Y · المقاعد المتاحة: Z".
- **الحضور** Attendance: filter by group, change status, delete.
- **وسائل الدفع** Payment methods: add, activate/deactivate.



12. Community — `/community` (`src/routes/community.tsx`)

Approved posts, filter chips by type and category, title search. Link to publish.

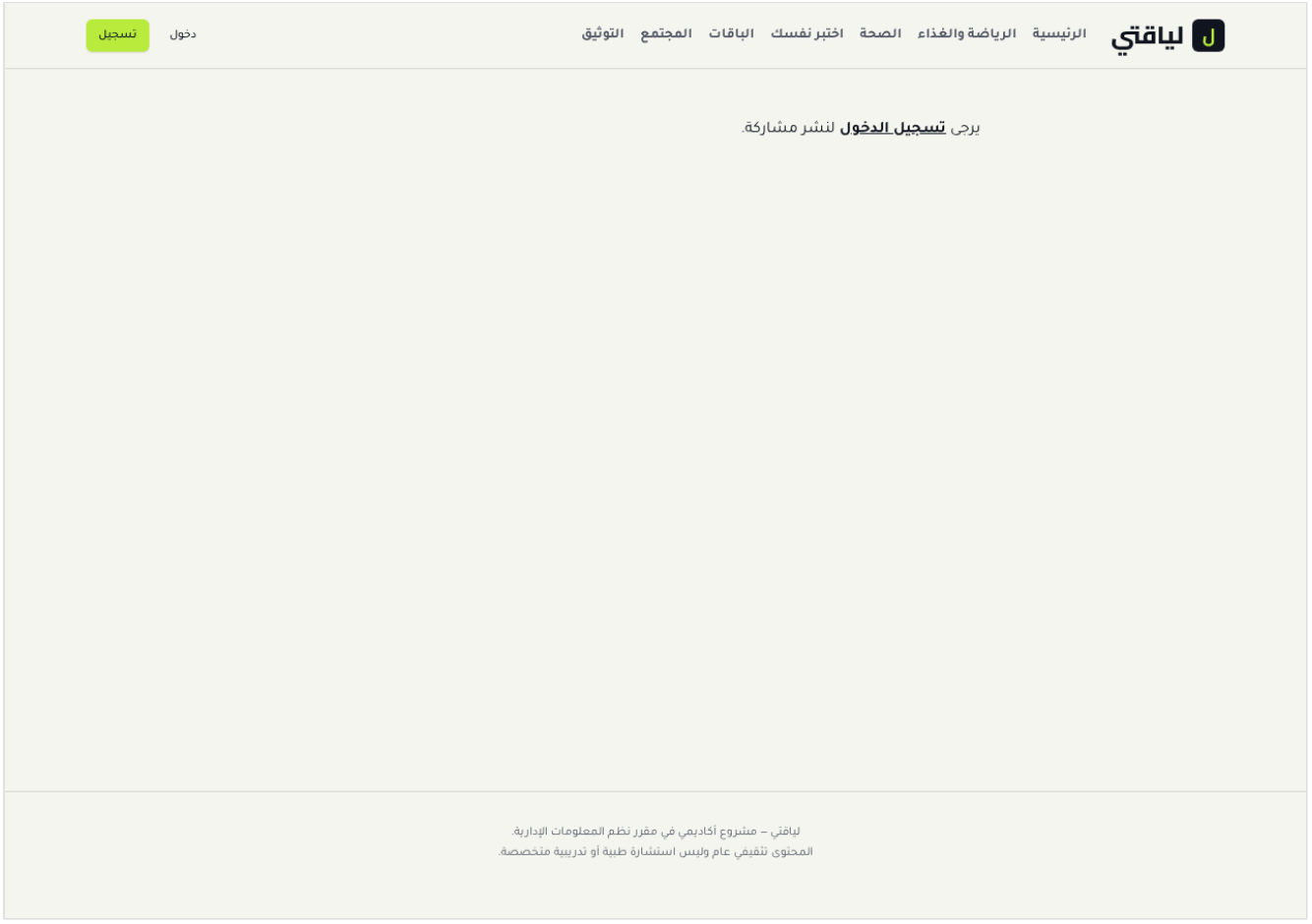


13. Post detail — `/posts/$id`

Post content, type badge, likes toggle, comments (add ≤ 2000, delete own; admin delete any), delete own post; approval status + reason visible to author/admin when not approved.

14. New post — `/new`

Inputs: type, category, title (3–150), content. Toast: "تم إرسال المشاركة، وستظهر للجميع بعد موافقة المشرف".



15. Profile — `/profile`

Inputs: display name (≥ 2, ≤ 50), bio (≤ 300), brief (≤ 300), fitness level, phone (≤ 20), age (clamped 10–100). Writes `profiles` and upserts `member_contacts`.

16. Documentation — `/about`

Arabic requirements, use cases, activity flow, class model and ERD as text diagrams.

مشروع أكاديمي – نظم المعلومات الإدارية

توثيق المشروع

لياقتي موقع لنشر معلومات اللياقة البدنية والتغذية الرياضية بشكل آمن، ويضم ثلاثة أنواع من المستخدمين: المشرف، والمدرب الرياضي المتخصص، والمشترك. تلخص هذه الصفحة التحليل والتصميم الذي بُني عليه النظام.

١. المشكلة والأهداف

كثير من الإصابات ناتجة عن أداء التمارين بشكل خاطئ وغياب الإرشاد، مع ضعف الوعي بالتغذية الرياضية المناسبة.

- مساعدة المستخدمين في الحصول على معلومات صحيحة عن التمارين.
- توفير معلومات عن التغذية الرياضية.
- نشر الثقافة الرياضية وتقديم محتوى تثقيفي للأعضاء.
- توفير أدوات لمتابعة الصحة واللياقة.

٢. تحليل المتطلبات

المتطلبات الوظيفية

- م.و1 – التسجيل (الاسم، الجوال، العمر، البريد، كلمة المرور) وتسجيل الدخول والتحقق من البريد.
- م.و2 – المشرف يدير المشتركين: قبول/رفض طلبات الاشتراك، الحظر.
- م.و3 – المشرف يدير المدربين المتخصصين: إضافة، حظر، حذف الصلاحية.
- م.و4 – المشرف يضيف مجموعات الاشتراك ووسائل الدفع/المشاركة.
- م.و5 – المدرب يكتب إرشادات التمارين والأجهزة مع المزايا والعيوب.
- م.و6 – المدرب يضيف منشورات، يراجع الحضور، يرسل تنبيهات للمشتركين ويجيب عن الأسئلة.
- م.و7 – المشترك يختار مجموعة ويدفع للخدمة (بانتظار تأكيد المشرف).
- م.و8 – المشترك يسجّل حضوره ويحدد التمارين المنفذة.
- م.و9 – المشترك يطّلع على الإرشادات، ينشر، يعلّق، ويتواصل بطرح الأسئلة.
- م.و10 – الجميع يحدّثون بياناتهم الشخصية، ويستخدمون صفحتي الصحة واختبر نفسك.

المتطلبات غير الوظيفية

05 — User Workflows

Each workflow reflects the actual code path. Diagrams are in `diagrams/` (see 09).

1. Registration and login

1. Visitor opens "تسجيل" → `/auth?mode=up` and enters name, email, password, optional phone/age.
2. Client validates with Zod; on success calls `supabase.auth.signUp` with metadata.
3. Database trigger `handle_new_user` creates `profiles`, `member_contacts`, a `subscriber` role, and `admin` if no admin exists yet.
4. User is sent to `/verify`; can resend the link.
5. After confirming, user logs in at `/auth?mode=in` → `signInWithPassword`.
6. `AuthProvider` loads roles and block status → redirect to `/dashboard`. Diagram: `diagrams/activity-authentication.png`.

2. Subscriber: group registration (capacity)

1. Open `/plans` (or dashboard attendance card when not in a group).
2. If `subscriber_count >= capacity` → "لا يوجد مكان في المجموعة"; stop.
3. Otherwise "اشترك الآن" → `/checkout/$id`; choose a payment method and optional note.
4. Client re-checks blocked status, existing non-rejected subscription (→ "أنت مسجّل بالفعل في هذه المجموعة") and fresh capacity.
5. Insert `subscriptions` (pending). Trigger `check_group_space` locks the group row and raises `ALREADY_REGISTERED` or `NO_SPACE_IN_GROUP` if violated.
6. Trigger `after_subscription_change` recounts the seat total (`sync_group_count`, counting non-rejected rows).
7. Insert `payments` (pending) with the group price.
8. Admin → Requests: **accept** (subscription active, payments confirmed) or **reject** (both rejected; seat freed). The database also lets admins delete a subscription (which frees the seat), but the admin page has no delete button for subscriptions. Diagram: `diagrams/activity-group-registration.png`.

3. Attendance

1. Subscriber opens Dashboard → Attendance. Without an active group, the available groups with seats are shown.
2. With an active group: select group, enter exercises performed, press record.
3. Client blocks if already recorded today for that group.
4. Database policy requires own row, `attend_date = today`, status `recorded`, no note, active member; trigger `validate_attendance` requires active subscription in that group, date not in the future and within group dates, and one record per subscriber/group/day.
5. Specialist → Attendance review: set `reviewed` or `rejected` with an optional note.
6. Admin → Attendance: filter by group, change status, delete.
7. Subscriber can delete own record only while `recorded`. Diagram: `diagrams/activity-attendance.png`.

4. Specialist content submission

1. Specialist opens `/exercises`, fills `GuideForm` (title, category, guidance, instructions, advantages, disadvantages, optional group).
2. Insert into `devices`; policy requires `specialist_id = me` and specialist role.
3. Trigger `enforce_content_approval` sets `pending`. Toast: "تم إرسال الإرشاد للمشرف، وسيُنشر بعد موافقته".
4. The item is visible to the author (badge "بانتظار موافقة المشرف") and admins only. Posts by any non-admin member follow the same path via `/new`.

5. Admin review, approval and rejection

1. Admin → "مراجعة المحتوى" lists pending guidance and posts.

2. **Approve** → `approval_status = approved`; item becomes public.
3. **Reject** → browser prompt for a reason (required, ≤ 500) → `approval_status = rejected`, `rejection_reason` saved.
4. Rejected items stay hidden from everyone except author and admin, who see status and reason. Diagrams: `activity-content-approval.png`, `sequence-content-review.png`, `state-content.png`.

6. Resubmission

1. Author edits a rejected (or approved) item.
2. Trigger detects a content change and sets `pending`, clearing `rejection_reason`.
3. Item re-enters the admin review queue.

7. Questions and alerts

1. Subscriber asks in Dashboard → Questions (≥ 3 chars, ≤ 2000).
2. Specialist sees unanswered questions first and answers (sets `answer_text`, `specialist_id`, `answer_date`).
3. Specialist sends an alert (2–1000 chars) to a subscriber; the subscriber sees it in Alerts and marks it read.

8. User management

1. Admin → Users lists profiles with contacts and roles.
2. "Set as trainer" inserts / removes the `specialist` role.
3. "Block" sets `profiles.status = blocked`; blocked members can still sign in and read, but every write policy using `is_active_member()` refuses them, and pages show "حسابك محظور".
4. Members cannot change their own `status` (trigger `protect_profile_status`).

9. Community participation

Browse `/community` → open post → like/unlike, comment, delete own comment/post. Publish via `/new` (goes to review).

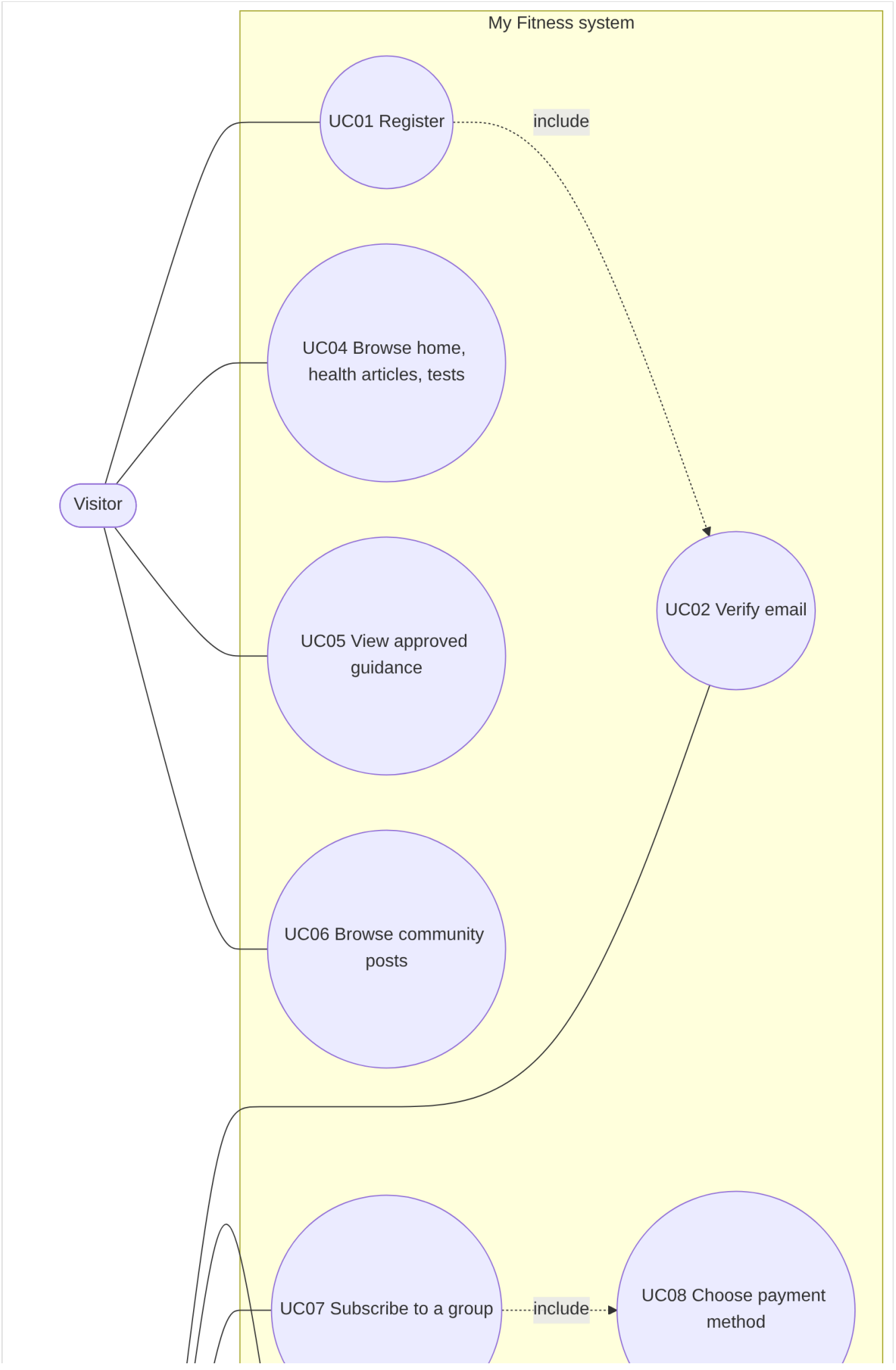
06 — Use Case Model

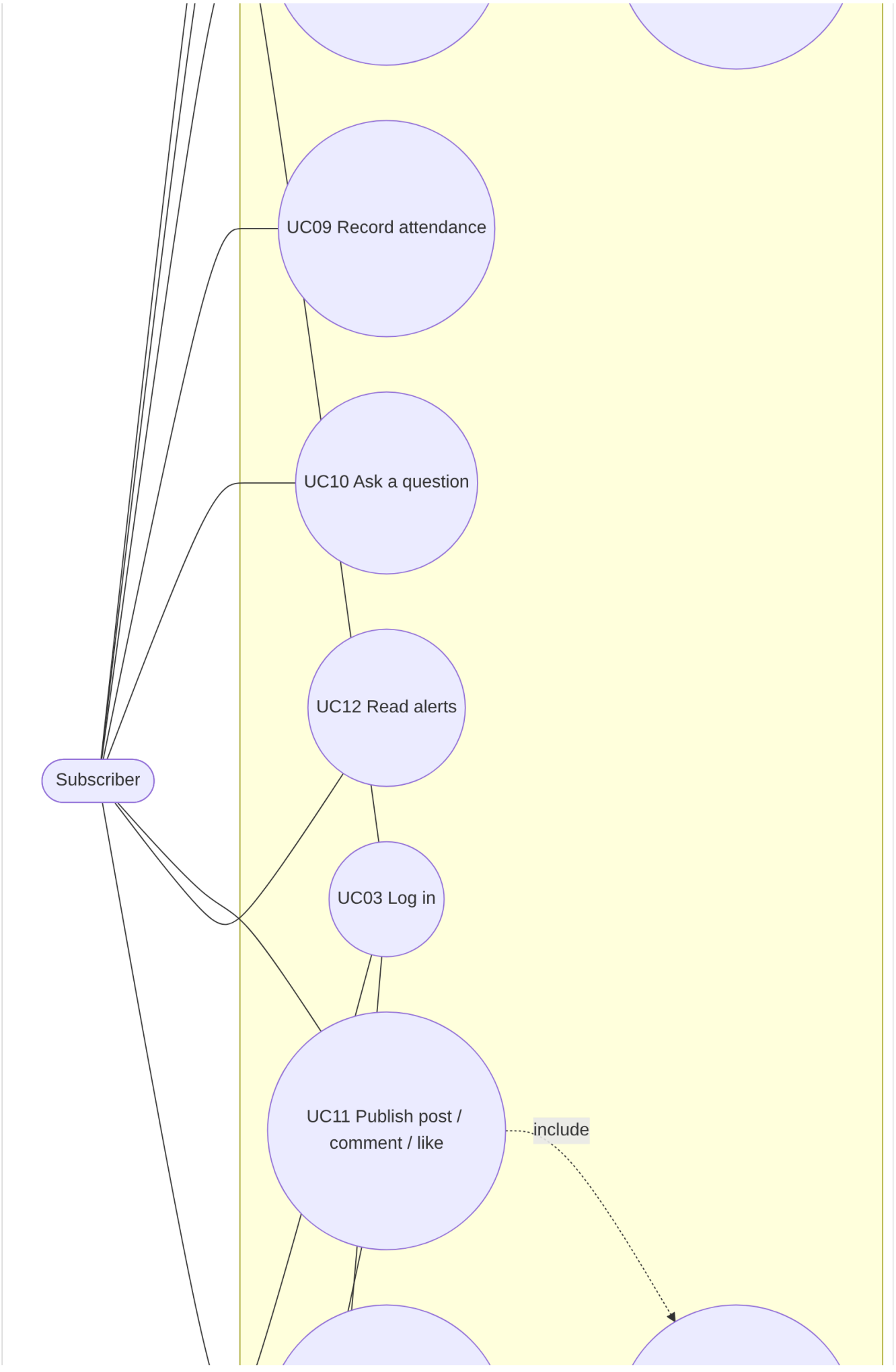
Actors

Actor	Description
Visitor	Unauthenticated user
Subscriber	Registered trainee (role `subscriber`)
Specialist	Trainer (role `specialist`), also a subscriber account
Admin	Platform administrator (role `admin`)
System (secondary)	Database triggers enforcing rules

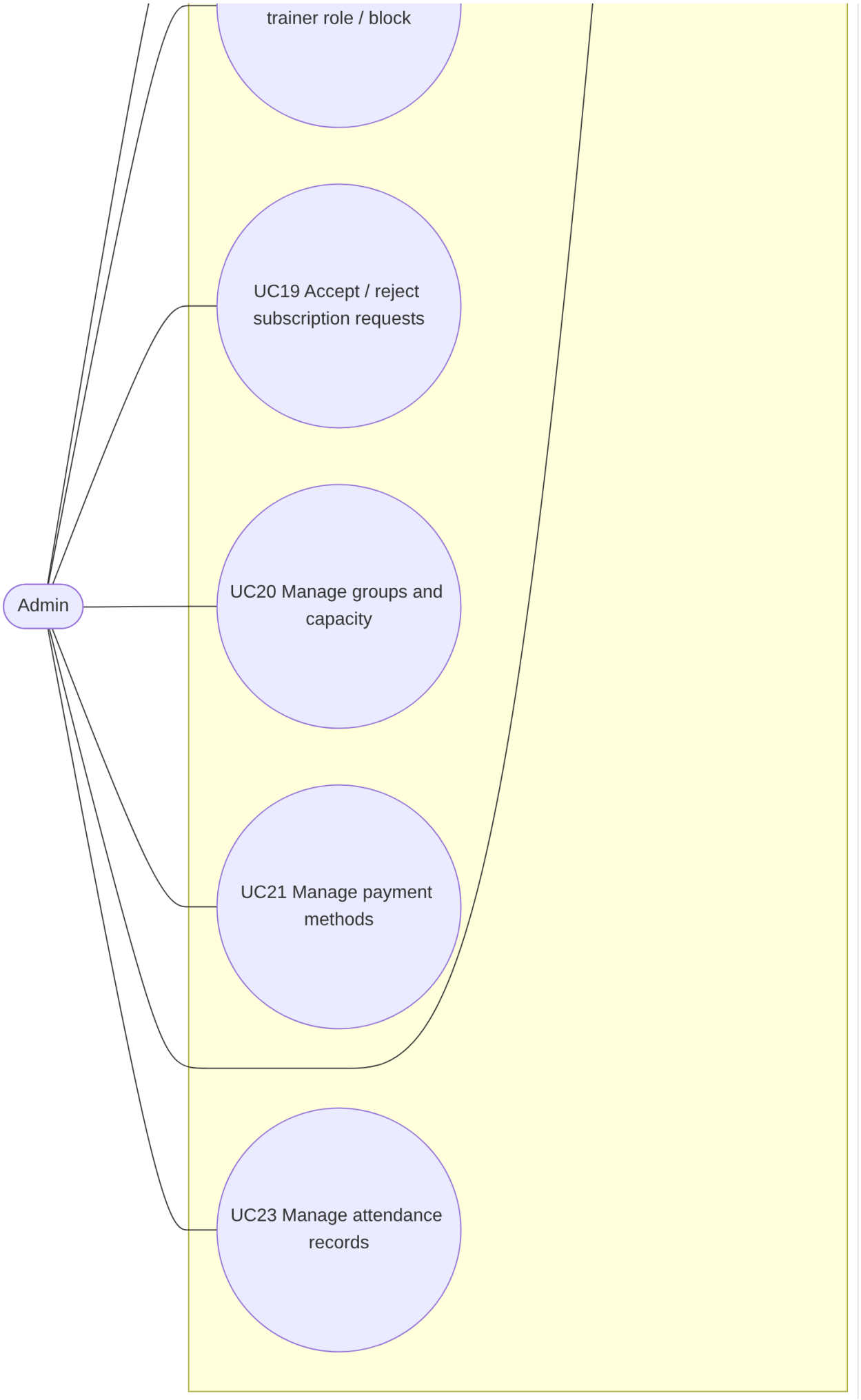
Use case diagram

Source: `diagrams/use-case.mmd`


My Fitness system
UC01 Register
include
UC04 Browse home, health articles, tests
Visitor
UC05 View approved guidance
UC02 Verify email
UC06 Browse community posts
UC07 Subscribe to a group
include
UC08 Choose payment method


UC09 Record attendance
UC10 Ask a question
UC12 Read alerts
Subscriber
UC03 Log in
UC11 Publish post / comment / like
include


UC14 Write device/exercise guidance
include
UC22 Review content: approve / reject
Specialist / Trainer
UC15 Answer questions
UC13 Manage own profile
UC16 Review attendance
UC17 Send alert
UC18 Manage users:


trainer role / block
UC19 Accept / reject subscription requests
Admin
UC20 Manage groups and capacity
UC21 Manage payment methods
UC23 Manage attendance records

Relationships: «include» Register → Verify email; Subscribe → Choose payment method; Write guidance → Review content; Publish post → Review content. Generalisation between Specialist and Subscriber exists only in the sense that a specialist account is also a subscriber account (both roles assigned); it is not modelled as inheritance in code.

Use case list

UC01 Register · UC02 Verify email · UC03 Log in · UC04 Browse home/health/tests · UC05 View approved guidance · UC06 Browse community · UC07 Subscribe to a group · UC08 Choose payment method · UC09 Record attendance · UC10 Ask a question · UC11 Publish post/comment/like · UC12 Read alerts · UC13 Manage own profile · UC14 Write guidance · UC15 Answer questions · UC16 Review attendance · UC17 Send alert · UC18 Manage users · UC19 Accept/reject subscription requests · UC20 Manage groups and capacity · UC21 Manage payment methods · UC22 Review content · UC23 Manage attendance records

Detailed descriptions

UC01 — Register

- **Actor:** Visitor · **Pre:** not signed in.
- **Main flow:** 1) open registration 2) enter data 3) system validates 4) system creates account and sends confirmation 5) system shows Verify page.
- **Alternative:** A1 optional phone/age omitted → stored as null.
- **Exception:** E1 invalid input → error toast; E2 email already used → backend error message.
- **Post:** auth user, profile, contact and subscriber role exist (admin role if first user).

UC03 — Log in

- **Actor:** any registered user · **Pre:** account exists.
- **Main:** enter email/password → session created → dashboard.
- **Alternative:** A1 email not confirmed → Verify page.
- **Exception:** E1 wrong credentials → "البريد الإلكتروني أو كلمة المرور غير صحيحة".
- **Post:** session active; roles loaded.

UC07 — Subscribe to a group

- **Actor:** Subscriber · **Pre:** signed in, not blocked, group active.
- **Main:** 1) choose group 2) system checks space 3) choose payment method (UC08) 4) submit 5) system checks duplicate & capacity 6) creates pending subscription and payment 7) increments count.
- **Alternative:** A1 group without capacity → no seat limit.
- **Exception:** E1 group full → "لا يوجد مكان في المجموعة"; E2 already registered → "أنت مسجّل بالفعل في هذه المجموعة"; E3 blocked → "حسابك محظور من قبل المشرف".
- **Post:** subscription pending, seat reserved until admin decision.

UC09 — Record attendance

- **Actor:** Subscriber · **Pre:** active subscription in the group; today within group dates.
- **Main:** select group → enter exercises → record → status `recorded`.
- **Alternative:** A1 single active group → selected automatically.
- **Exception:** E1 already recorded today; E2 not active in group; E3 outside group period; E4 blocked.
- **Post:** one attendance row for today.

UC10 — Ask a question / UC15 — Answer questions

- **Actors:** Subscriber / Specialist · **Pre:** signed in; not blocked (asking).
- **Main:** subscriber submits text → specialist sees it (unanswered first) → writes answer → subscriber sees answer.
- **Exception:** text too short → error.
- **Post:** query with `answer_text`, `specialist_id`, `answer_date`.

UC14 — Write guidance

- **Actor:** Specialist · **Pre:** has specialist role.
- **Main:** fill form → submit → saved as pending → UC22.
- **Alternative:** A1 edit own guidance → returns to pending. A2 delete own guidance.
- **Exception:** E1 missing title/guidance → error; E2 non-specialist → refused by database.
- **Post:** guidance pending review.

UC16 — Review attendance

- **Actor:** Specialist (also Admin via UC23) · **Main:** open list → approve (`reviewed`) or reject (`rejected`) with note. **Post:** status updated.

UC17 — Send alert

- **Actor:** Specialist · **Main:** choose subscriber, write message (2–1000) → saved. **Post:** alert unread for subscriber.

UC18 — Manage users

- **Actor:** Admin · **Main:** view users → toggle trainer role or block/unblock. **Exception:** database error shown as toast. **Post:** roles/status updated.

UC19 — Accept/reject subscription requests

- **Actor:** Admin · **Pre:** pending request. **Main:** accept → subscription active, payments confirmed. **Alternative:** reject → both rejected, seat freed. **Post:** seat count recalculated.

UC20 — Manage groups and capacity

- **Actor:** Admin · **Main:** create group with details and optional capacity; toggle active; update capacity.
- **Exception:** capacity ≤ 0 → error; capacity below current subscribers → "لا يمكن أن تكون السعة أقل من عدد المشتركين الحاليين".

UC21 — Manage payment methods

- **Actor:** Admin · **Main:** add name/description; activate/deactivate. Inactive methods are hidden from checkout.

UC22 — Review content

- **Actor:** Admin · **Pre:** pending post or guidance.
- **Main:** open Content review → approve → public.
- **Alternative:** reject → enter reason → hidden, author sees reason.
- **Exception:** empty reason → rejection cancelled.
- **Post:** `approval_status` approved/rejected.

UC23 — Manage attendance records

- **Actor:** Admin · **Main:** filter by group → change status or delete.

UC11 — Publish post / comment / like

- **Actor:** Subscriber · **Pre:** signed in, not blocked (post/comment).
- **Main:** write post → pending → UC22; comment/like on approved posts.
- **Post:** content stored; post public after approval.

07 — Object-Oriented Analysis and Design

Approach

The analysis identified domain objects from the presentation's use cases and ERD. In the implementation, these objects are realised as **database entities** (tables) with behaviour split between (a) database functions/triggers and (b) React components that act on them. The code is written in a functional React style: there are **no application-level TypeScript classes**. The class diagram below is therefore a **domain (analysis) class model** that corresponds one-to-one to real tables; the listed operations are real responsibilities implemented by the named trigger, policy or component.

Classes

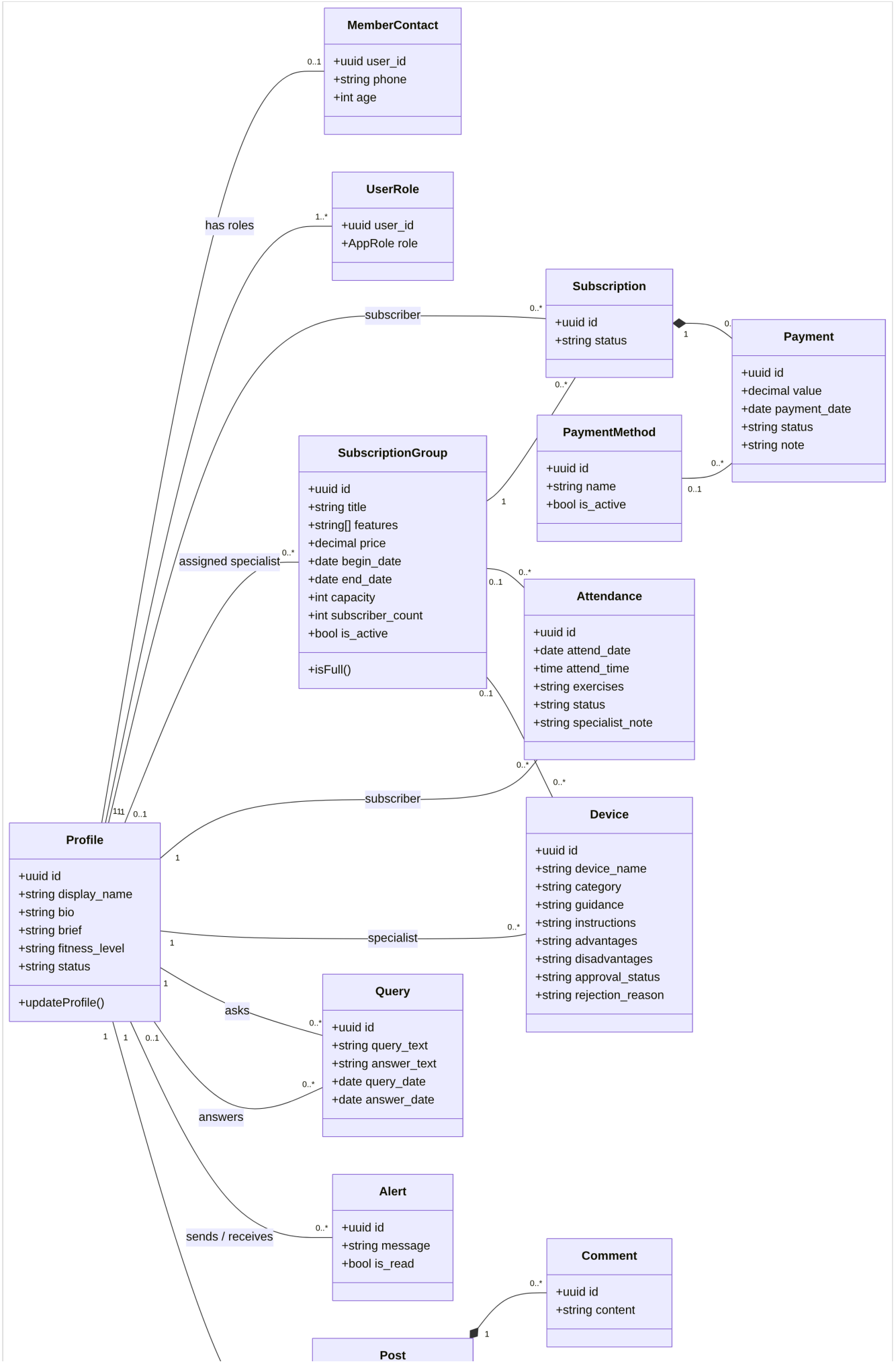
Class (table)	Key attributes	Responsibilities / operations (where implemented)
Profile (`profiles`)	id, display_name, bio, brief, fitness_level, status	Update own profile (`profile.tsx`); status protected (`protect_profile_status`); `is_active_member()`
MemberContact (`member_contacts`)	user_id, phone, age	Private contact data, readable by owner/admin
UserRole (`user_roles`)	user_id, role	`has_role()`; admin assigns specialist (`admin.tsx` Users)
SubscriptionGroup (`subscription_groups`)	title, features, price, dates, capacity, subscriber_count, is_active, specialist_id	`isFull()` (`lib/fitness.ts`); `sync_group_count()`; admin CRUD
Subscription (`subscriptions`)	subscriber_id, group_id, status	`check_group_space()`; `after_subscription_change()`; admin accept/reject
Payment (`payments`)	subscription_id, method, value, date, status, note	Created with subscription; admin confirm/reject
PaymentMethod (`payment_methods`)	name, description, is_active	Admin add/toggle
Attendance (`attendance`)	subscriber_id, group_id, date, time, exercises, status, specialist_note	`validate_attendance()`; record, review, manage
Device / Guidance (`devices`)	device_name, category, guidance, instructions, advantages, disadvantages, group_id, approval_status, rejection_reason	`enforce_content_approval()`; `GuideForm` add/edit
Query (`queries`)	query_text, answer_text, dates, specialist_id	Ask (`MyQueries`), answer (`AnswerQueries`)
Alert (`alerts`)	message, is_read	Send (trainer tab), mark read (`MyAlerts`)
Post (`posts`)	post_type, category, title, content, approval_status, rejection_reason	`enforce_content_approval()`; publish (`new.tsx`); `fetchPosts()`
Comment (`comments`)	content	Add/delete (`posts.$id.tsx`)
PostLike (`post_likes`)	post_id, user_id	Like/unlike

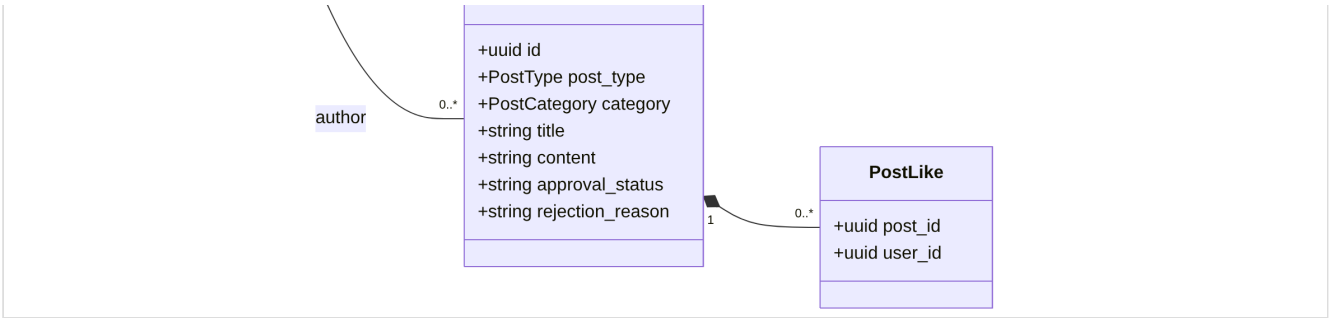
Relationships

- **Association:** Profile–Subscription–SubscriptionGroup (many-to-many resolved by Subscription); Profile–Attendance; Profile–Query (asker and answerer); Profile–Alert (sender and receiver); Profile–Device (author); SubscriptionGroup–Device (optional); SubscriptionGroup–Profile (assigned specialist, optional).
- **Composition** (cascade delete in the database): Post ◆— Comment, Post ◆— PostLike, Subscription ◆— Payment (`ON DELETE CASCADE`).
- **Aggregation / optional links:** SubscriptionGroup ◇— Attendance (`ON DELETE SET NULL`), PaymentMethod ◇— Payment.
- **Inheritance: not used.** Roles are modelled as data (`user_roles`), not subclasses, so a single account can hold several roles.

Class diagram

Source: `diagrams/class-diagram.mmd`


MemberContact
+uuid user_id
+string phone
+int age
UserRole
+uuid user_id
+AppRole role
Subscription
+uuid id
+string status
Payment
+uuid id
+decimal value
+date payment_date
+string status
+string note
PaymentMethod
+uuid id
+string name
+bool is_active
SubscriptionGroup
+uuid id
+string title
+string[] features
+decimal price
+date begin_date
+date end_date
+int capacity
+int subscriber_count
+bool is_active
+isFull()
Attendance
+uuid id
+date attend_date
+time attend_time
+string exercises
+string status
+string specialist_note
Device
+uuid id
+string device_name
+string category
+string guidance
+string instructions
+string advantages
+string disadvantages
+string approval_status
+string rejection_reason
Profile
+uuid id
+string display_name
+string bio
+string brief
+string fitness_level
+string status
+updateProfile()
Query
+uuid id
+string query_text
+string answer_text
+date query_date
+date answer_date
Alert
+uuid id
+string message
+bool is_read
Comment
+uuid id
+string content
Post
has roles
subscriber
assigned specialist
subscriber
specialist
asks
answers
sends / receives



Design decisions (recorded in `AGENTS.md`)

- Roles in a separate table checked by `has_role()` — avoids privilege escalation.
- Rules (capacity, one-per-group, attendance validity, approval) enforced by triggers so they hold even if the UI is bypassed.
- Public read / owner write data access directly from the browser client with row-level rules; no server functions.

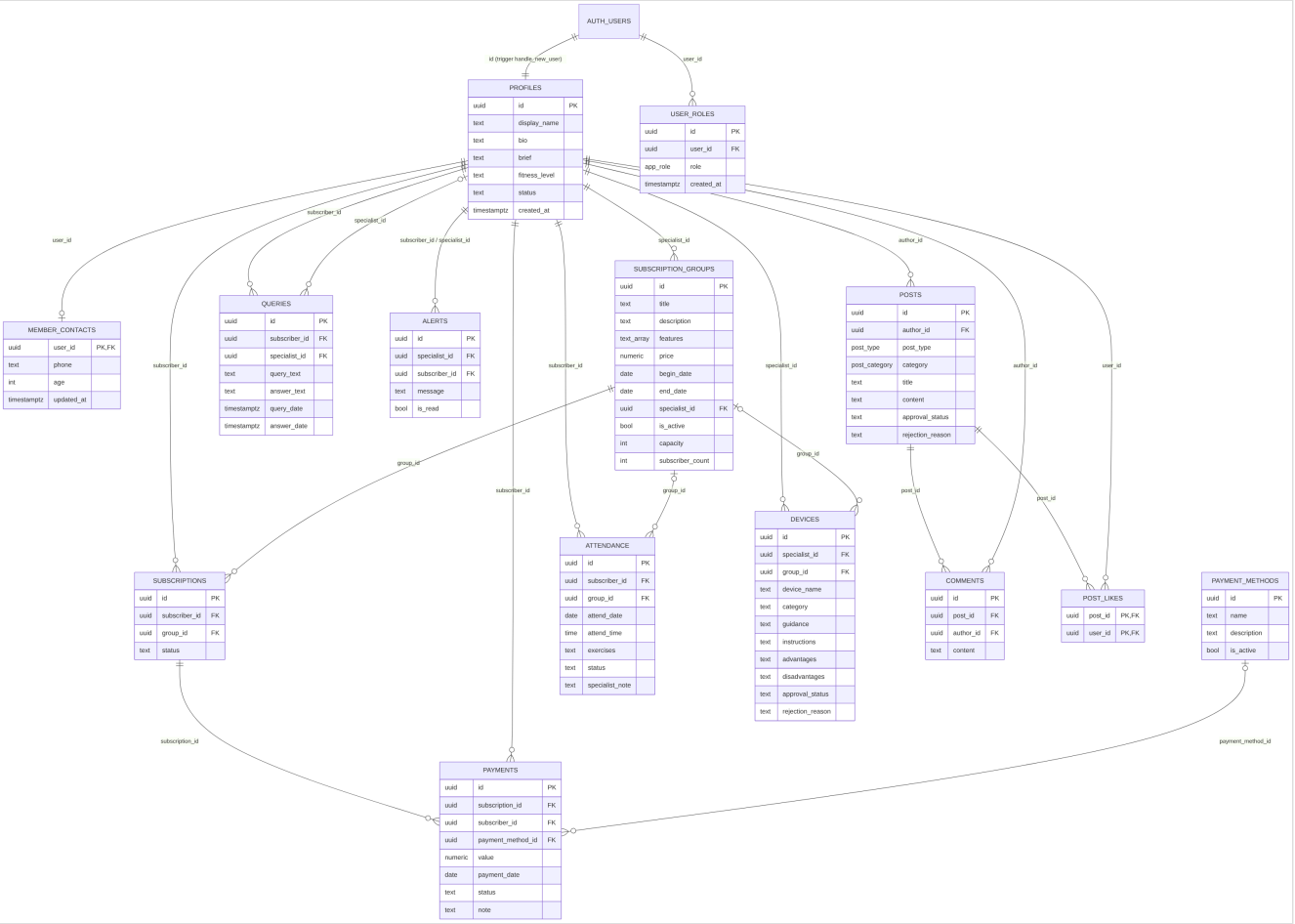
08 — Database Design

Overview

PostgreSQL database managed by Lovable Cloud. 14 tables in the `public` schema, 3 enum types, row-level security enabled on every table, and triggers enforcing business rules. Authentication accounts live in the managed `auth.users` table (not directly accessible to the app); `profiles.id` equals the auth user id and is created by the `handle_new_user` trigger. The complete generated column/constraint/policy listing is in **Appendix A**.

ERD

Source: `diagrams/erd.mmd`



Entities

Table	PK	Foreign keys	Purpose
`profiles`	id (= auth user id)	—	Public identity: name, bio, brief, fitness level, status (active/blocked)
`member_contacts`	user_id	→ profiles(id)	Private phone and age (5–120 check)
`user_roles`	id; unique(user_id, role)	→ auth.users(id) cascade	Role assignments
`subscription_groups`	id	specialist_id → profiles (set null)	Packages/groups; price ≥ 0; capacity; subscriber_count
`subscriptions`	id	subscriber_id → profiles; group_id → subscription_groups	Membership request; status pending/active/rejected

Table	PK	Foreign keys	Purpose
payments	id	subscription_id → subscriptions (cascade); subscriber_id → profiles; payment_method_id → payment_methods	Payment record; status pending/confirmed/rejected
payment_methods	id	—	Seeded: bank transfer, cash, cheque
attendance	id	subscriber_id → profiles; group_id → subscription_groups (set null)	Status recorded/reviewed/rejected; exercises 1–1000 chars
devices	id	specialist_id → profiles; group_id → subscription_groups	Trainer guidance; approval_status + rejection_reason
queries	id	subscriber_id, specialist_id → profiles	Questions and answers
alerts	id	specialist_id, subscriber_id → profiles (cascade)	Messages 2–1000 chars
posts	id	author_id → profiles (cascade)	Community posts; title 3–150, content 1–5000; approval
comments	id	post_id → posts, author_id → profiles (cascade)	Content 1–2000
post_likes	(post_id, user_id)	→ posts, → profiles (cascade)	One like per user per post

Enum types

- app_role: admin, specialist, subscriber
- post_type: challenge, solution, experience, tip
- post_category: strength, cardio, nutrition, flexibility, recovery, general

Business-rule triggers and functions

Function	Fires on	Rule
handle_new_user()	after insert on auth.users	Create profile, contact, subscriber role; first user → admin
has_role(uid, role)	used in policies	Security-definer role check
is_active_member(uid)	used in policies	True when profile status = active
protect_profile_status()	before update profiles	Only admins may change status
check_group_space()	before insert/update subscriptions	Raise ALREADY_REGISTERED / NO_SPACE_IN_GROUP
after_subscription_change() → sync_group_count()	after insert/update/delete subscriptions	Recount non-rejected subscriptions into subscriber_count
validate_attendance()	before insert/update attendance	Active in group, not future, within dates, one per day
enforce_content_approval()	before insert/update posts, devices	Non-admin → pending; edits reset to pending; admin inserts approved
update_updated_at_column()	before update (several tables)	Maintain updated_at

Seed data

- subscription_groups: "الباقة الفضية" (Silver, 50 SAR) and "الباقة البلاتينية" (Platinum, 60 SAR) with the presentation's features; no capacity set.
- payment_methods: حوالة مصرفية مباشرة (direct bank transfer), الدفع نقدًا (cash), مدفوعات الشيكات (cheque).
- Current record counts (27 Sep 2026): 0 users, 0 posts, 0 guidance items.

Mapping to the presentation's ERD

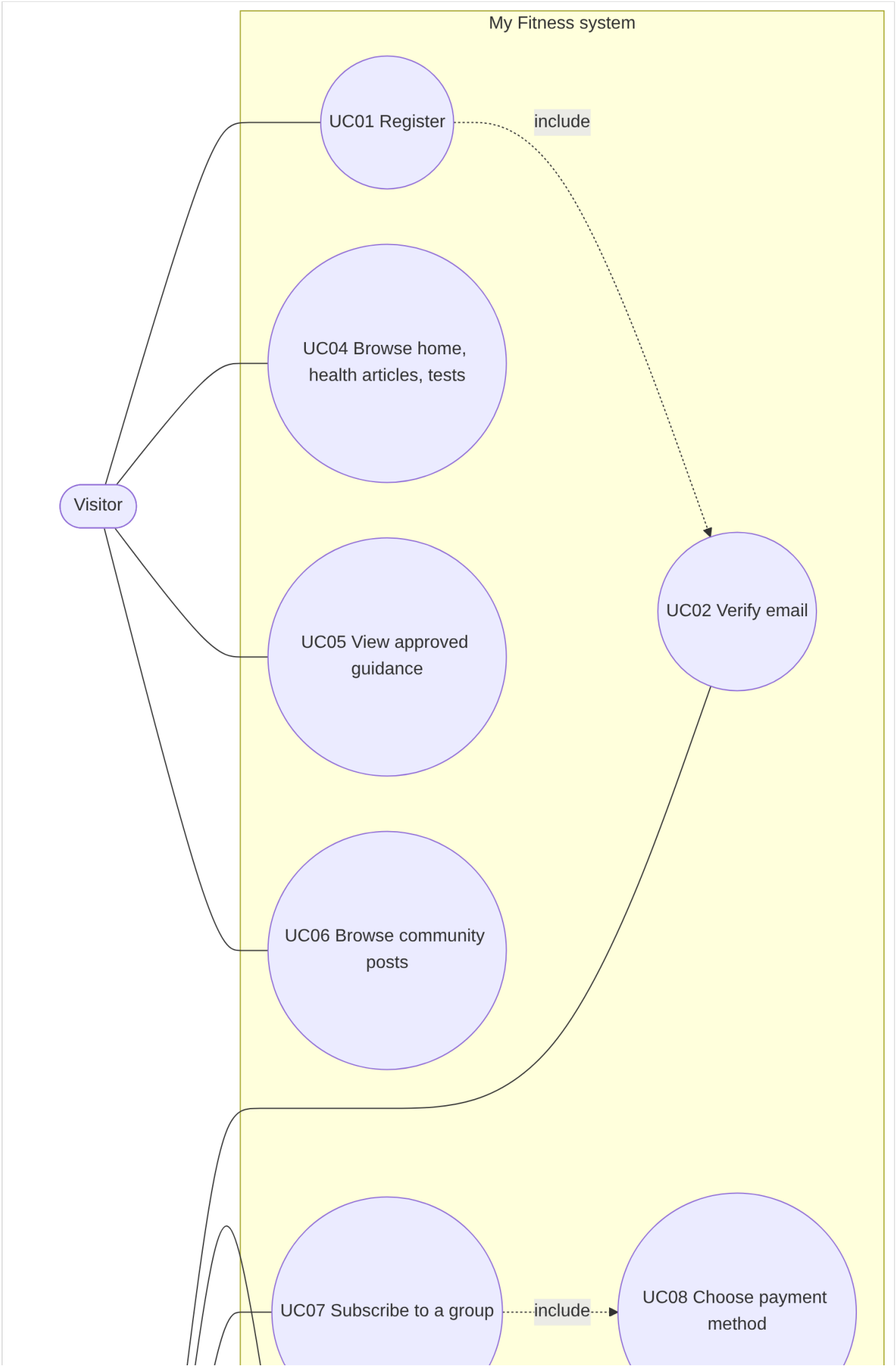
The presentation's entities Subscriber, Specialist and Admin are implemented as one profiles entity plus user_roles. Subscription group, subscription, payment, payment method, attendance, device (guidance), query and alert map directly. Posts/comments/likes implement the community "share challenges, solutions, experiences" objective.

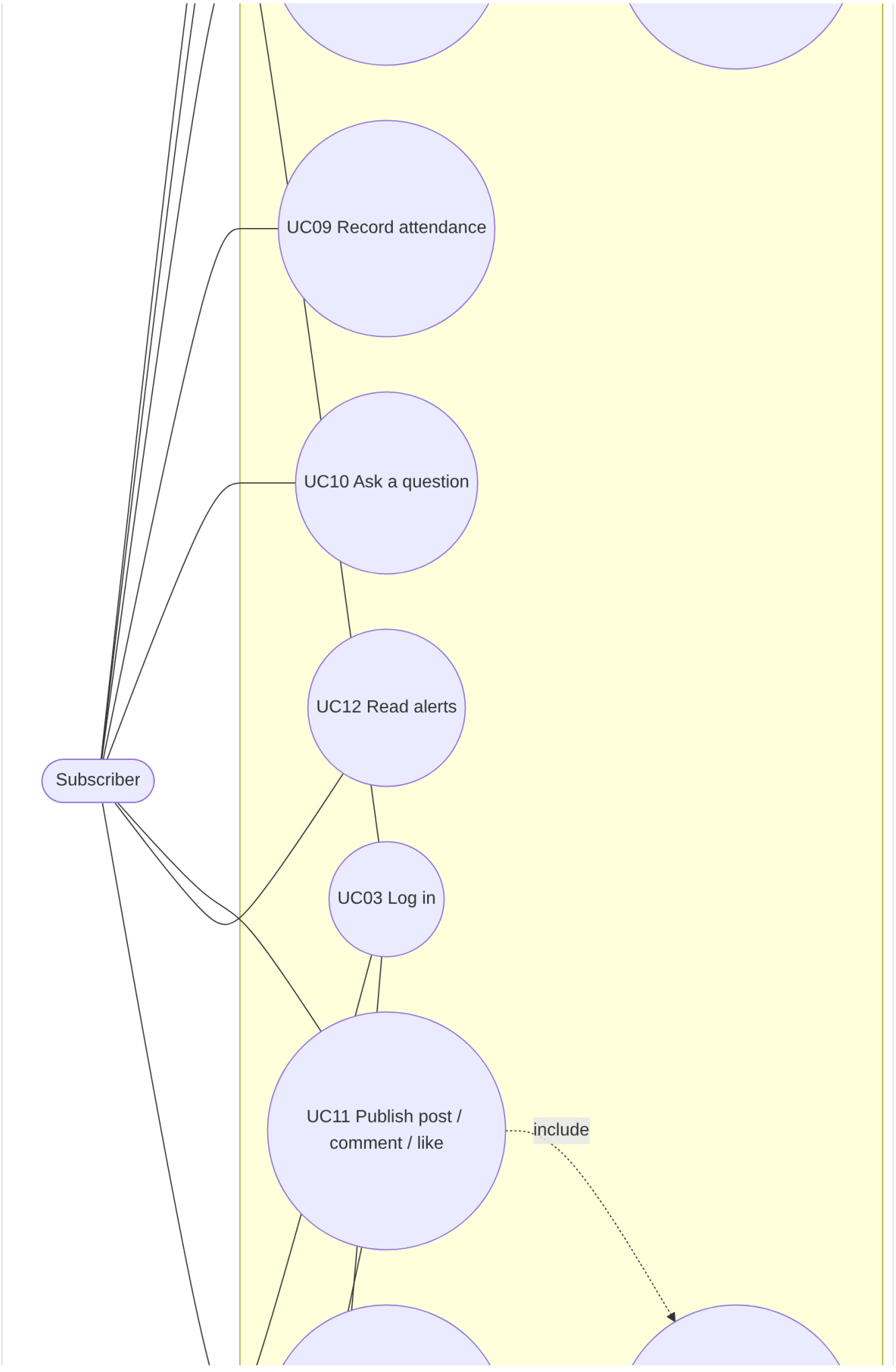
09 — UML Diagrams

All diagrams are written in Mermaid (`diagrams/*.mmd`) and rendered to `.svg` and `.png`. Each is derived from the real code and database.

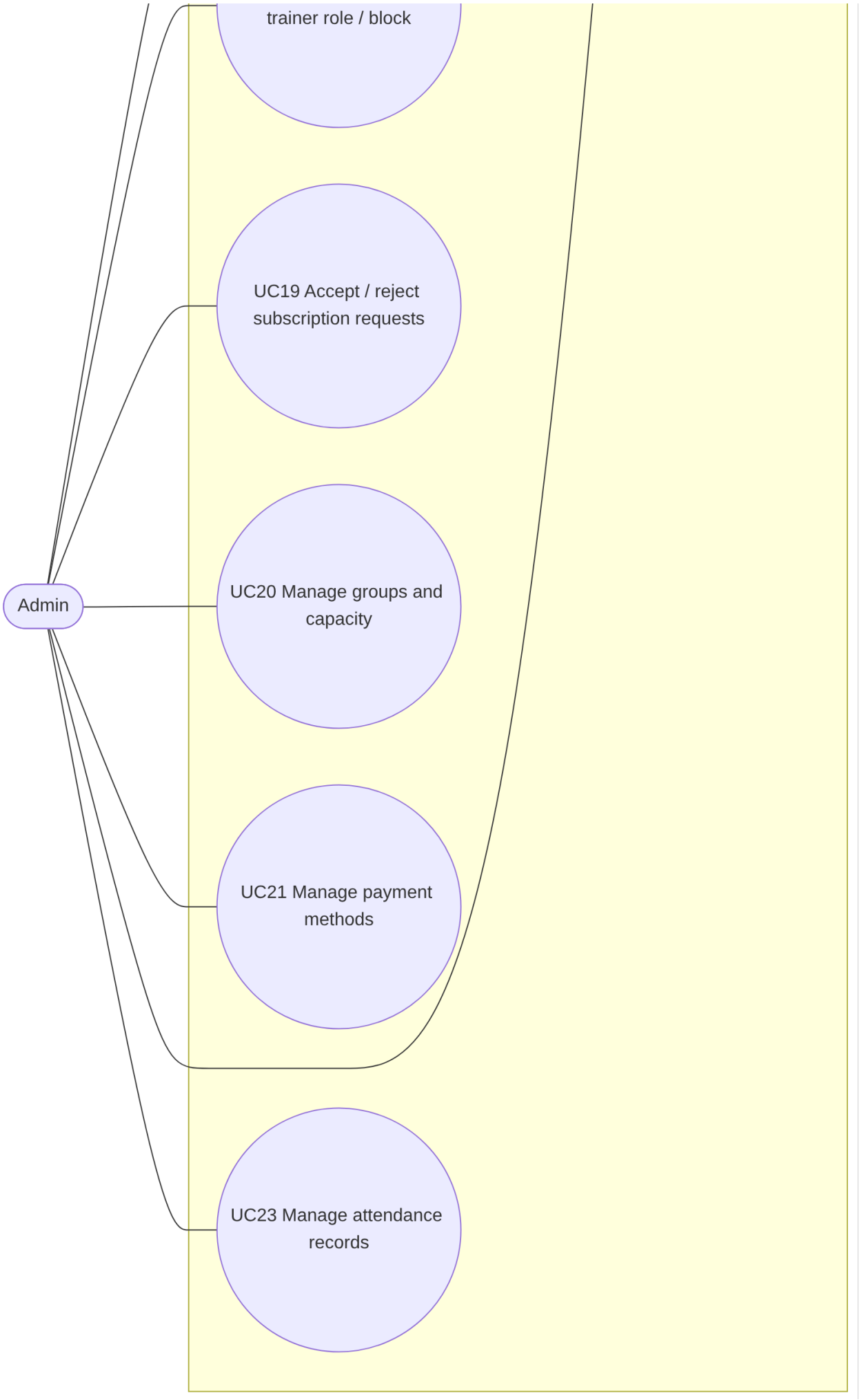
Diagram	Status	File
Use case	Provided	`use-case`
Class (domain)	Provided	`class-diagram`
ERD	Provided	`erd`
Activity — authentication	Provided	`activity-authentication`
Activity — content submission / review / approval / rejection / resubmission	Provided	`activity-content-approval`
Activity — group registration & capacity	Provided	`activity-group-registration`
Activity — attendance	Provided	`activity-attendance`
Sequence — subscribe to group	Provided	`sequence-subscribe`
Sequence — content submission & review	Provided	`sequence-content-review`
Sequence — attendance	Provided	`sequence-attendance`
State — content approval	Provided	`state-content`
State — subscription	Provided	`state-subscription`
State — attendance	Provided	`state-attendance`
Component	Provided	`component`
Deployment	Provided (high level)	`deployment`
Object diagram	Not applicable / insufficient project evidence (no live data)	—
Communication / timing diagrams	Not applicable	—

Use case

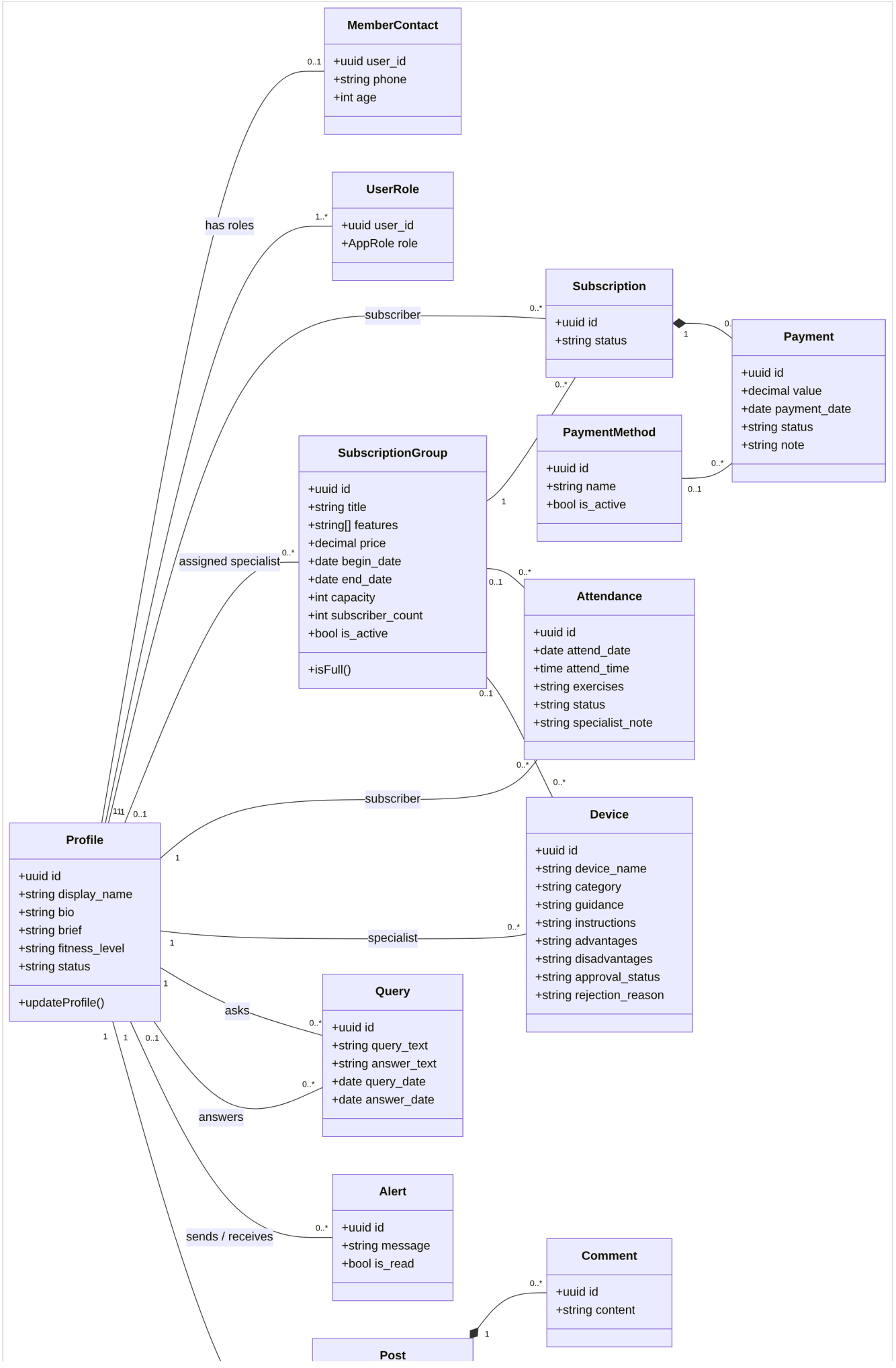

My Fitness system
UC01 Register
include
UC04 Browse home, health articles, tests
Visitor
UC05 View approved guidance
UC02 Verify email
UC06 Browse community posts
UC07 Subscribe to a group
include
UC08 Choose payment method

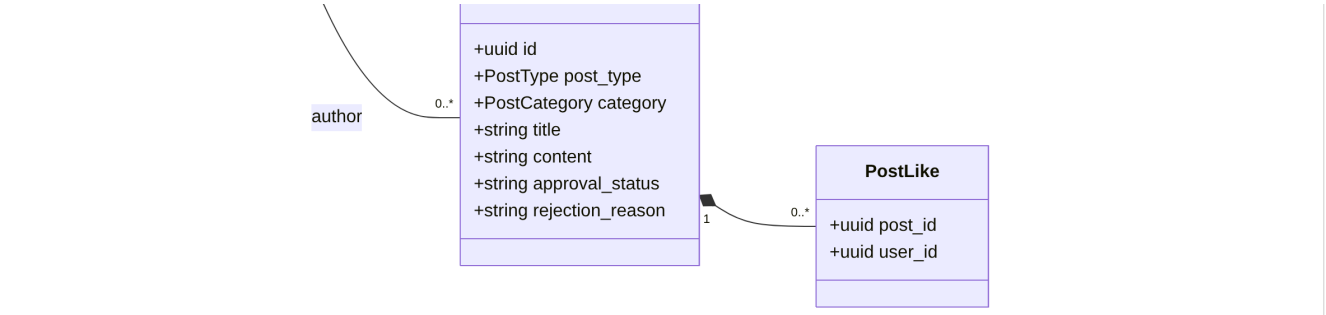

UC09 Record attendance
UC10 Ask a question
UC12 Read alerts
Subscriber
UC03 Log in
UC11 Publish post / comment / like
include


UC14 Write device/exercise guidance
include
UC22 Review content: approve / reject
Specialist / Trainer
UC15 Answer questions
UC13 Manage own profile
UC16 Review attendance
UC17 Send alert
UC18 Manage users:


trainer role / block
UC19 Accept / reject subscription requests
Admin
UC20 Manage groups and capacity
UC21 Manage payment methods
UC23 Manage attendance records

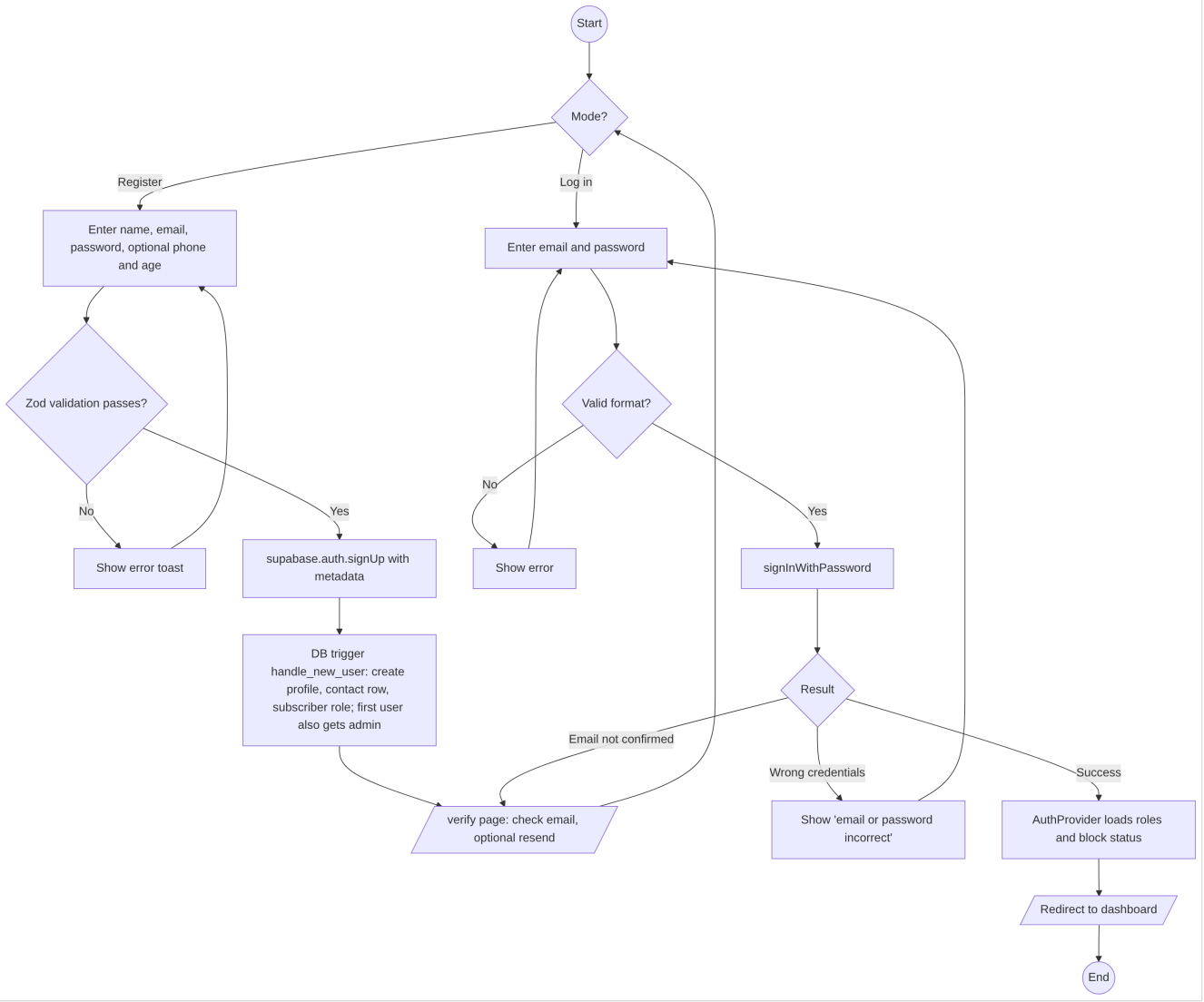
Class


MemberContact
+uuid user_id
+string phone
+int age
UserRole
+uuid user_id
+AppRole role
Subscription
+uuid id
+string status
Payment
+uuid id
+decimal value
+date payment_date
+string status
+string note
PaymentMethod
+uuid id
+string name
+bool is_active
SubscriptionGroup
+uuid id
+string title
+string[] features
+decimal price
+date begin_date
+date end_date
+int capacity
+int subscriber_count
+bool is_active
+isFull()
Attendance
+uuid id
+date attend_date
+time attend_time
+string exercises
+string status
+string specialist_note
Device
+uuid id
+string device_name
+string category
+string guidance
+string instructions
+string advantages
+string disadvantages
+string approval_status
+string rejection_reason
Profile
+uuid id
+string display_name
+string bio
+string brief
+string fitness_level
+string status
+updateProfile()
Query
+uuid id
+string query_text
+string answer_text
+date query_date
+date answer_date
Alert
+uuid id
+string message
+bool is_read
Comment
+uuid id
+string content
Post
has roles
subscriber
assigned specialist
subscriber
specialist
asks
answers
sends / receives

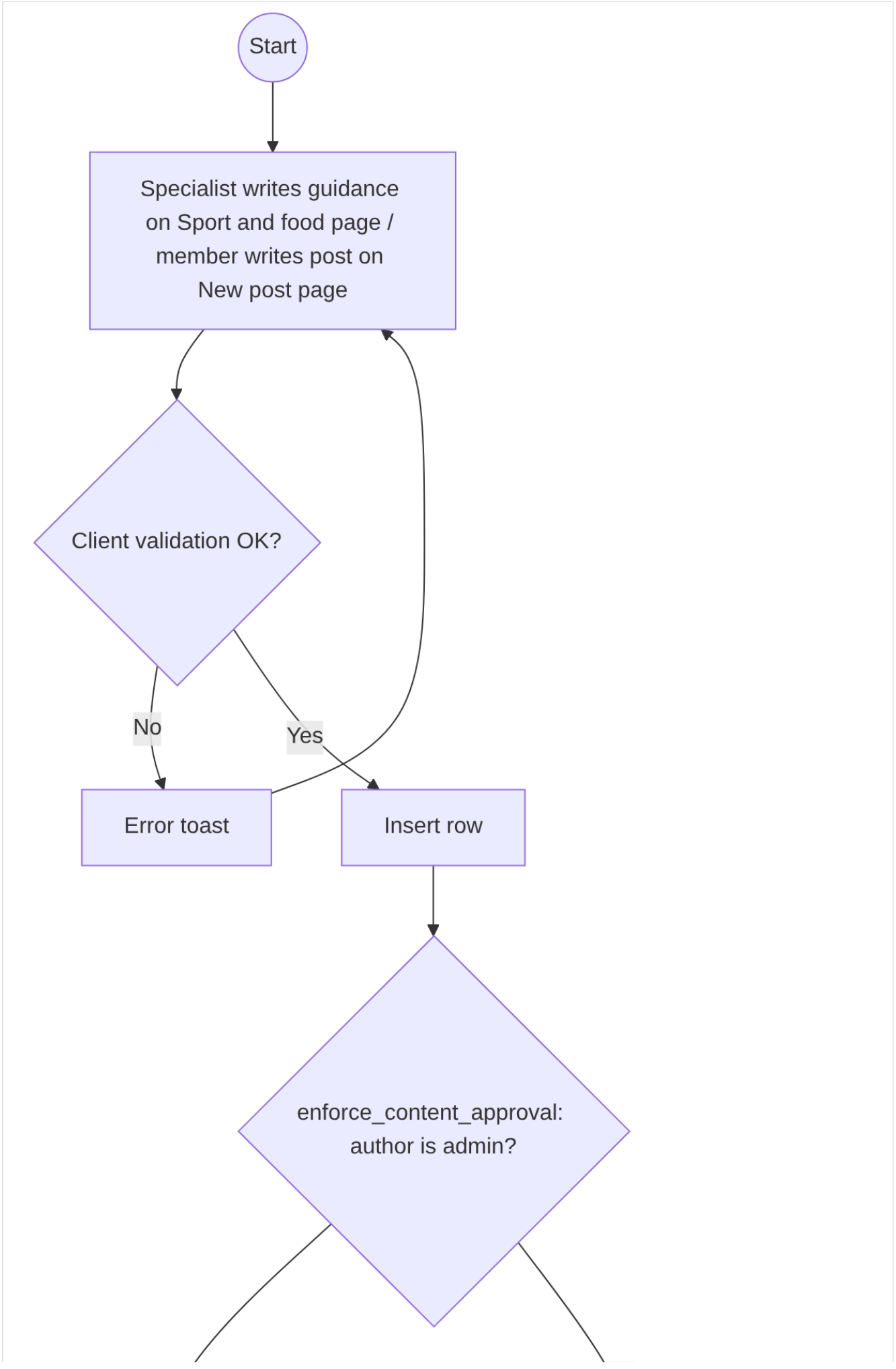


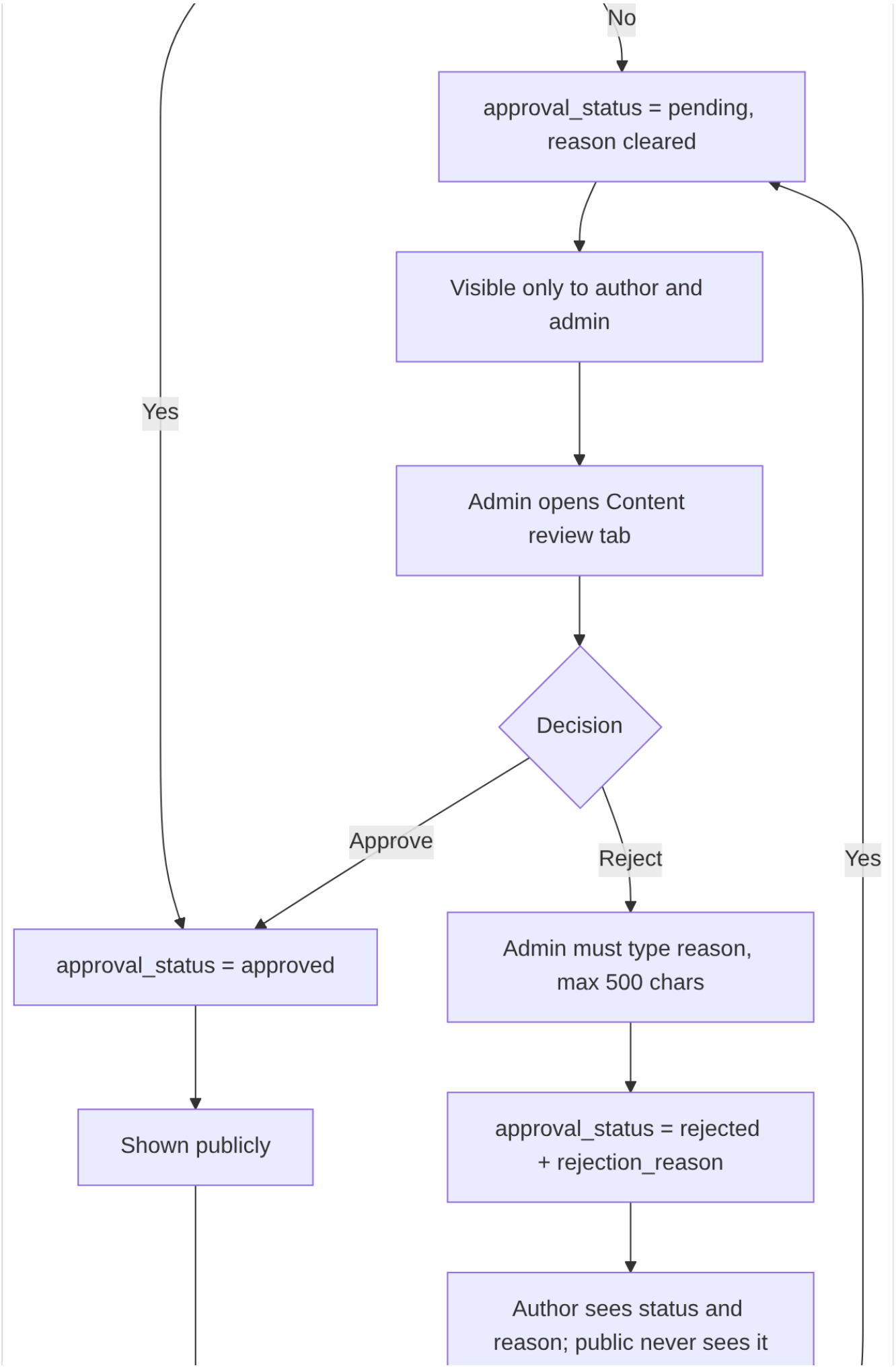
Activity diagrams

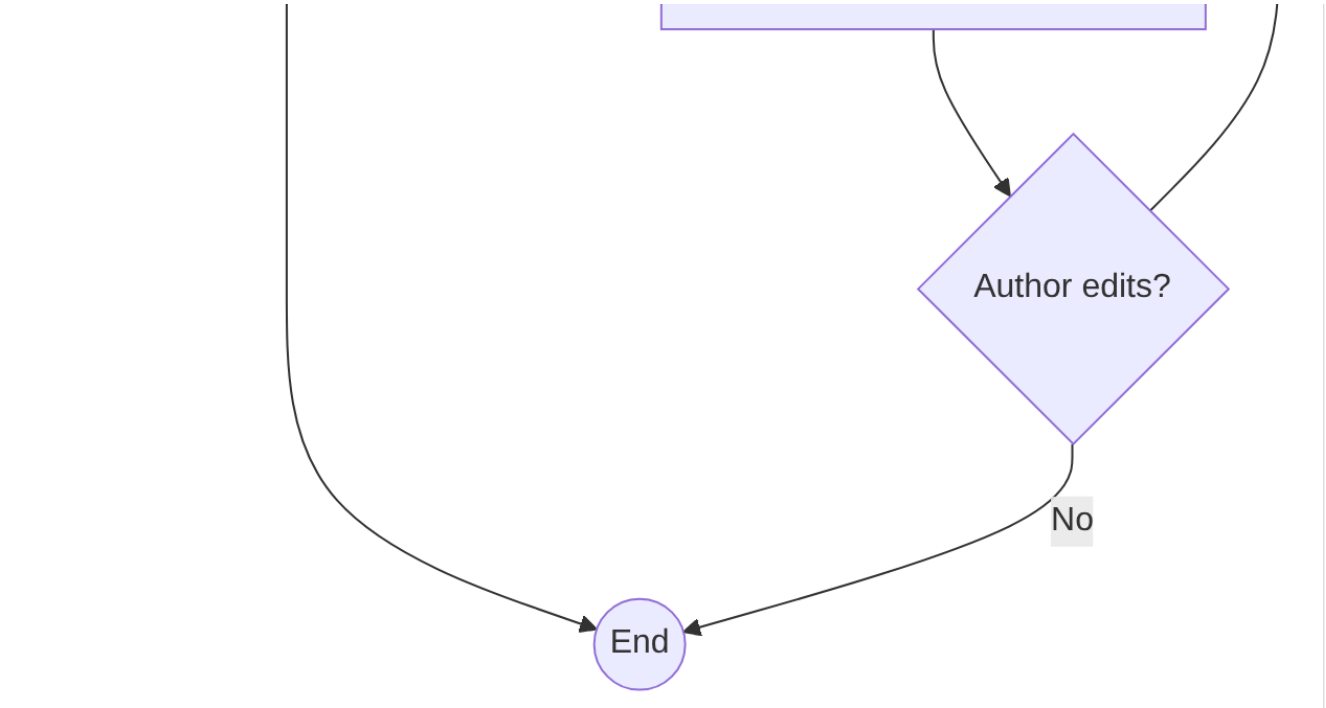
Authentication



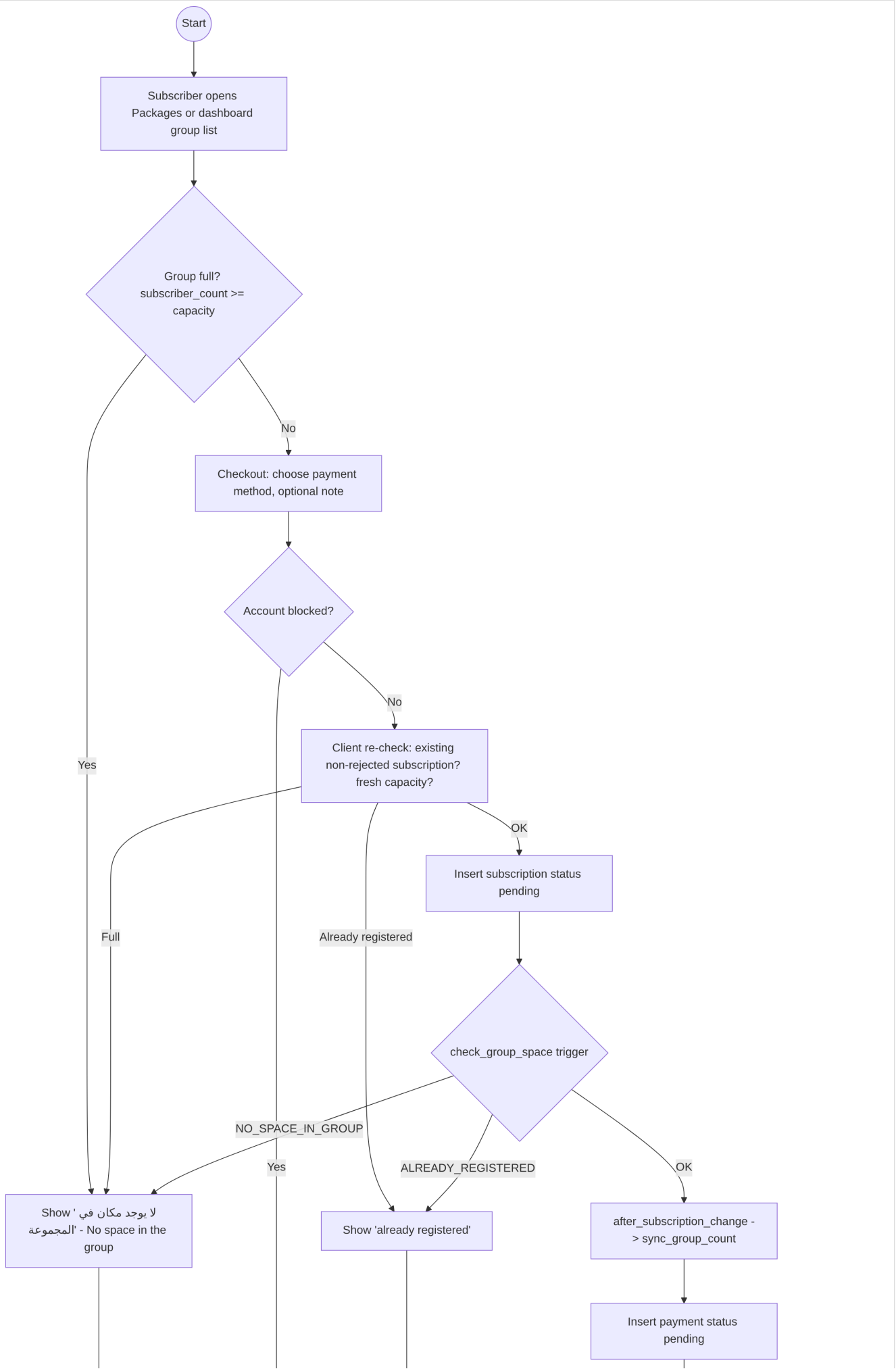
Content submission, review, approval, rejection, resubmission

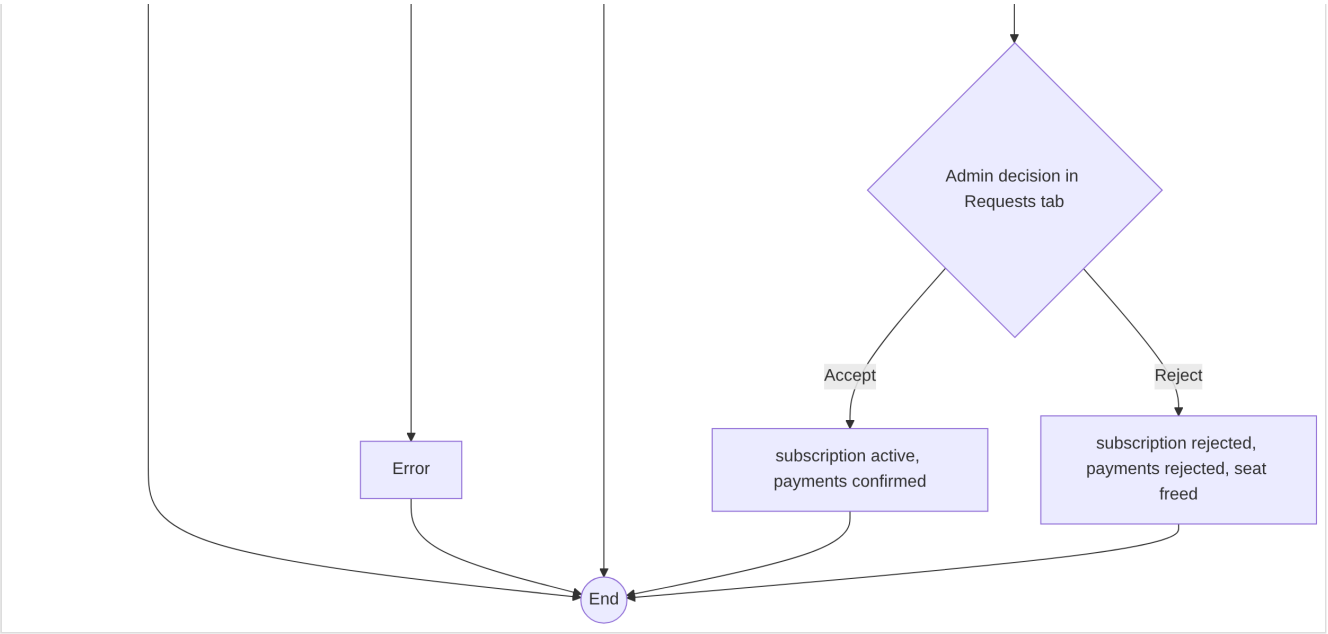

Start
Specialist writes guidance on Sport and food page / member writes post on New post page
Client validation OK?
No
Yes
Error toast
Insert row
enforce_content_approval: author is admin?


No
approval_status = pending, reason cleared
Visible only to author and admin
Admin opens Content review tab
Decision
Approve
Reject
Yes
Yes
approval_status = approved
Shown publicly
Admin must type reason, max 500 chars
approval_status = rejected + rejection_reason
Author sees status and reason; public never sees it

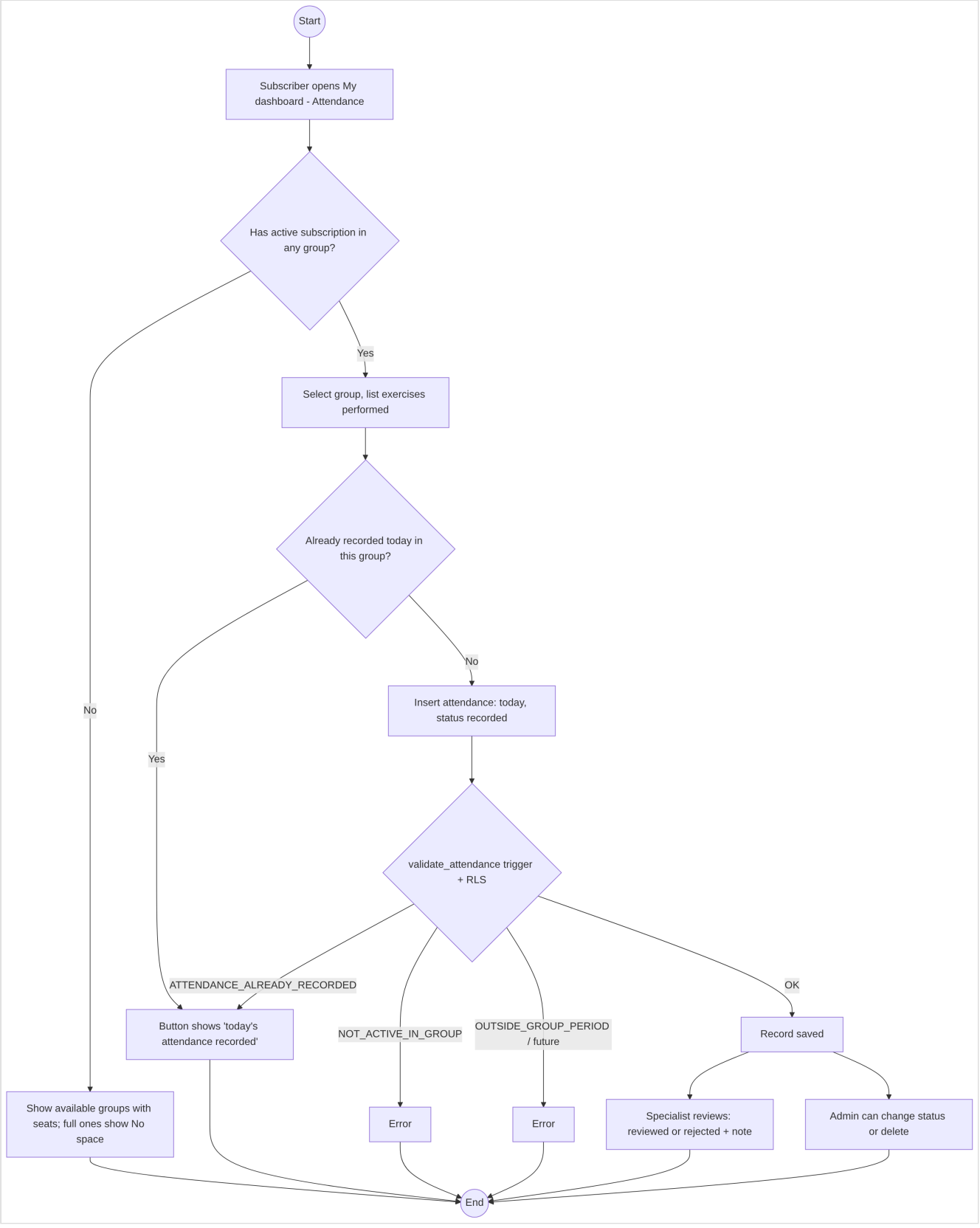


Group registration and capacity


Start
Subscriber opens Packages or dashboard group list
Group full? subscriber_count >= capacity
Yes
No
Checkout: choose payment method, optional note
Account blocked?
Yes
No
Client re-check: existing non-rejected subscription? fresh capacity?
Full
Already registered
OK
Insert subscription status pending
check_group_space trigger
NO_SPACE_IN_GROUP
Yes
ALREADY_REGISTERED
OK
Show ' لا يوجد مكان في المجموعة' - No space in the group
Show 'already registered'
after_subscription_change -> sync_group_count
Insert payment status pending


Admin decision in Requests tab
Accept
Reject
Error
subscription active, payments confirmed
subscription rejected, payments rejected, seat freed
End

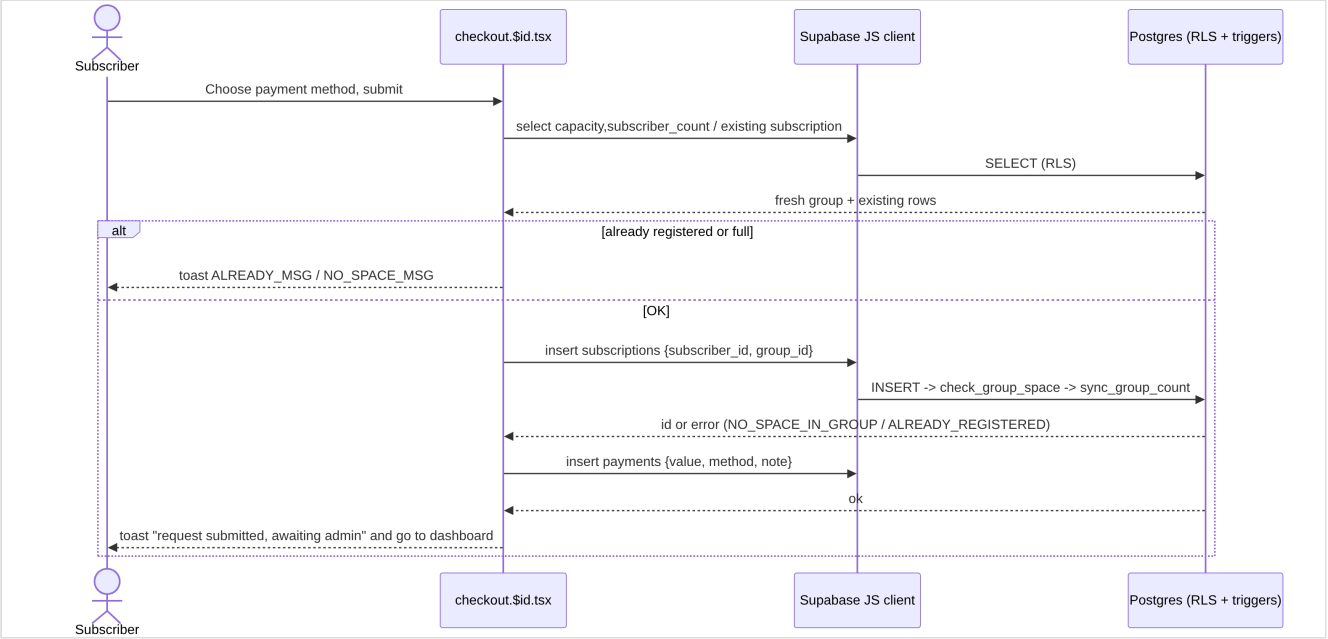
Attendance



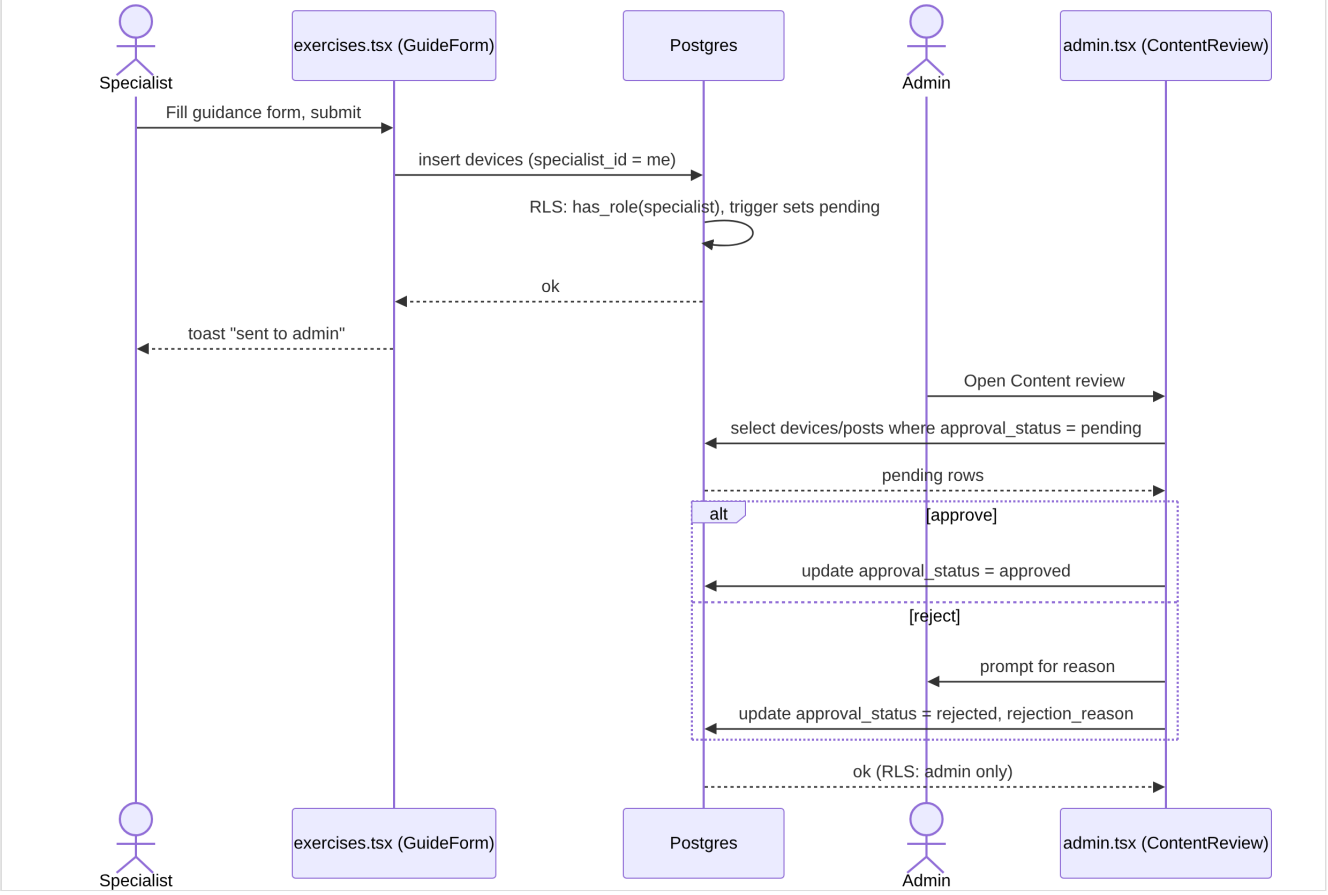
Sequence diagrams

Participants are the real React module, the Supabase JS client and the Postgres database; no other backend services exist.

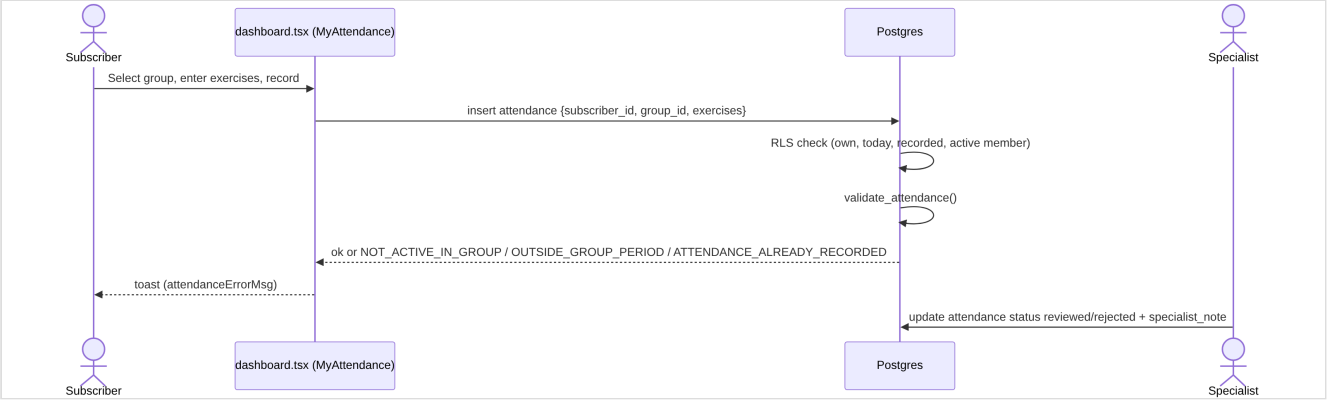
Subscribe to a group



Content submission and review

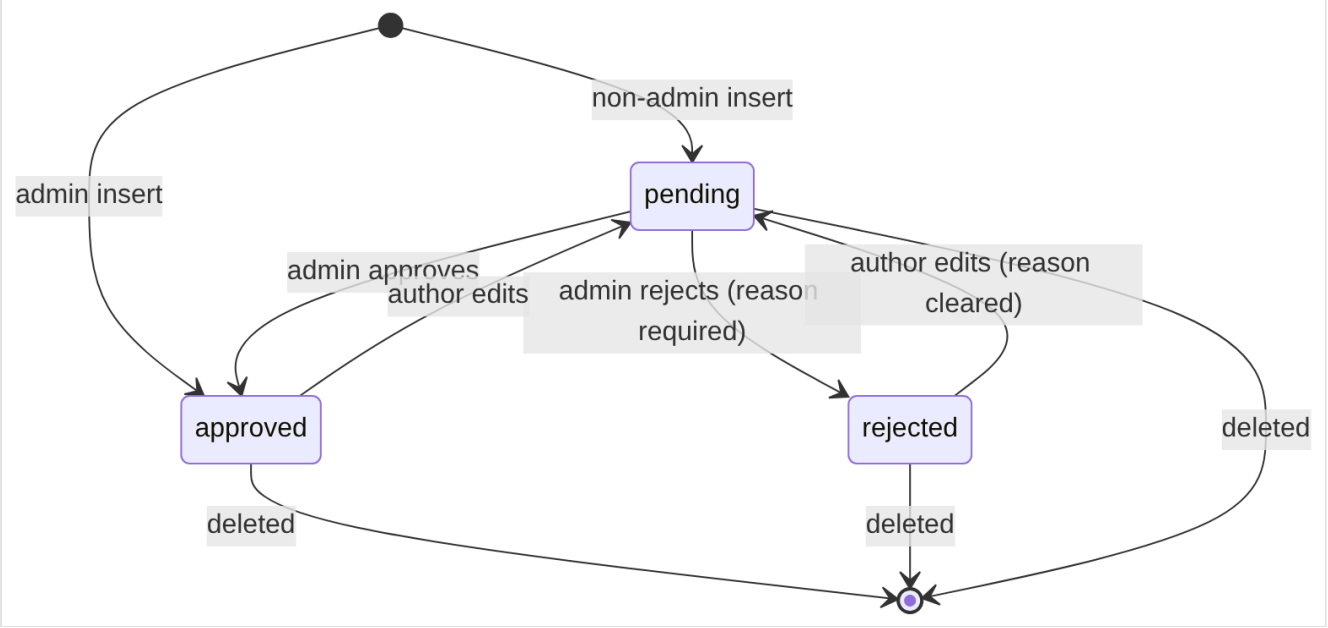


Attendance

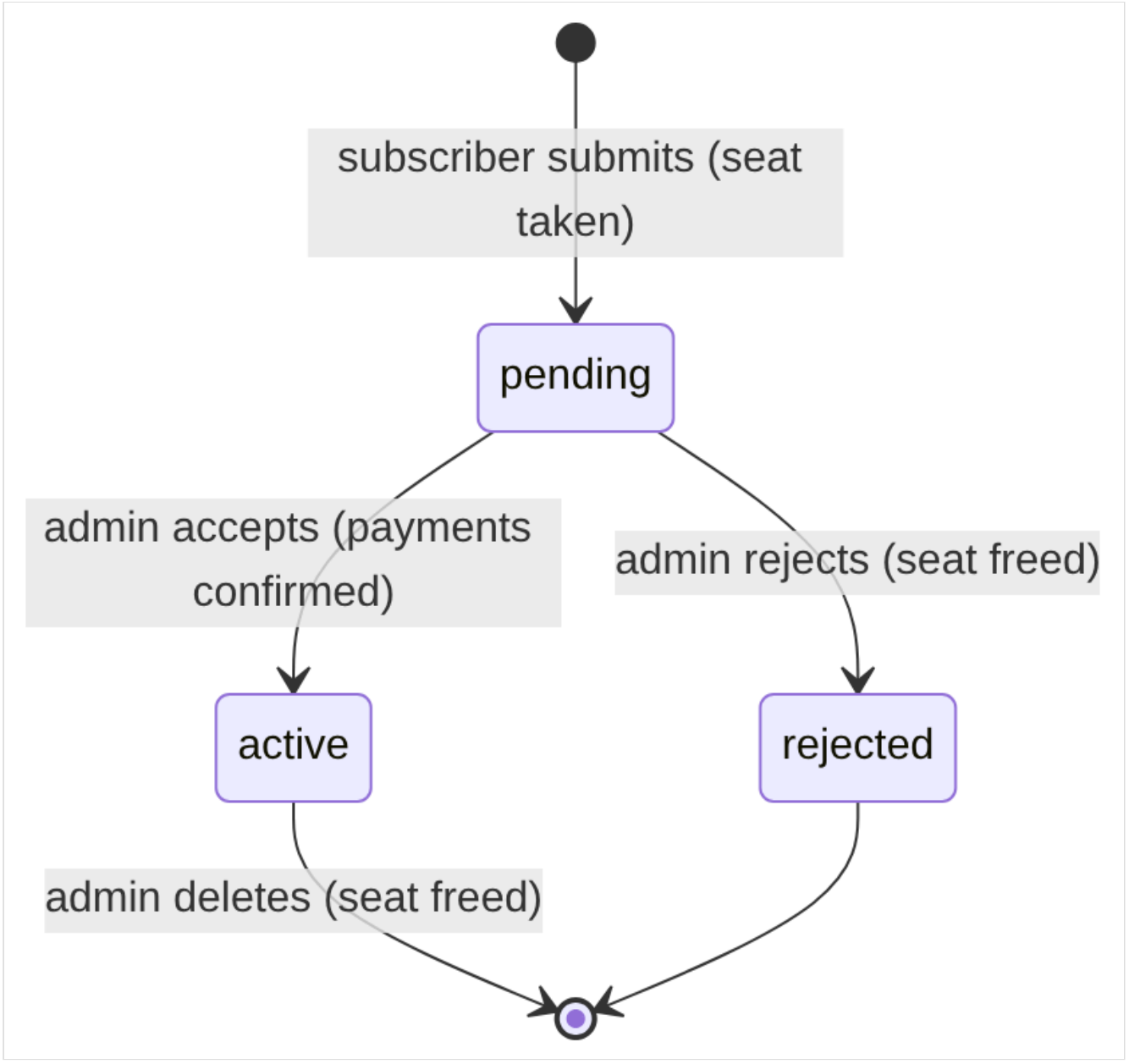


State diagrams

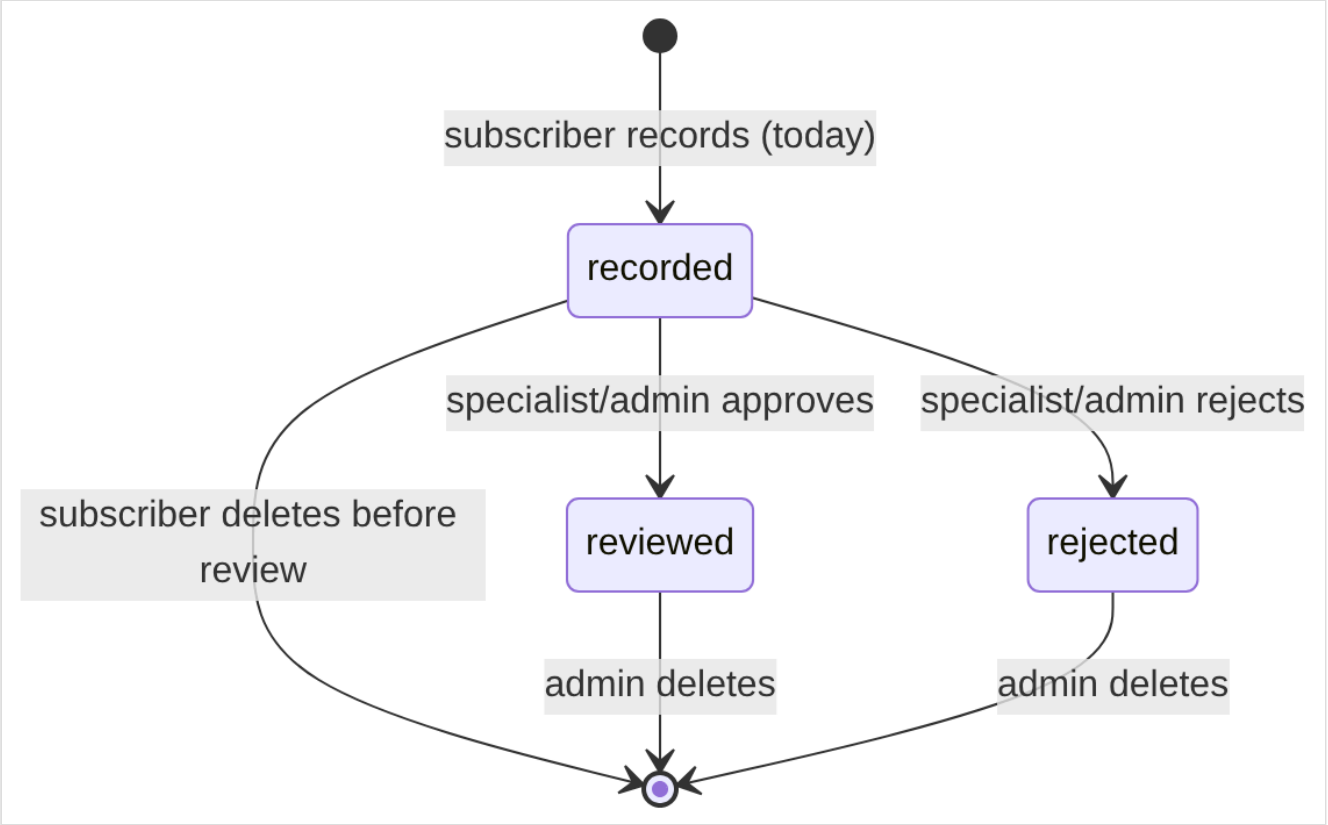
Content (`posts`, `devices`.approval_status)



Subscription (`subscriptions.status`)



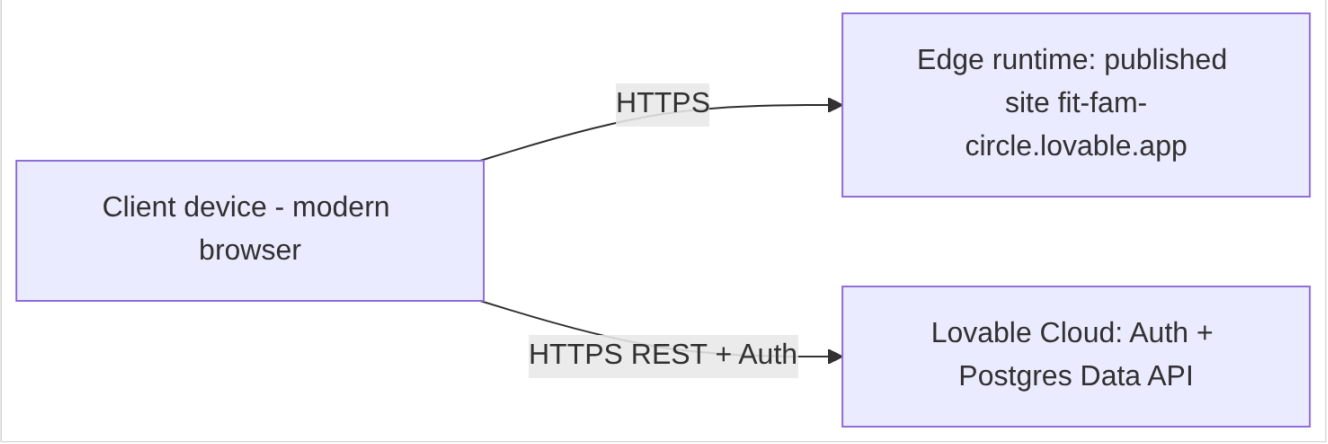
Attendance (`attendance.status`)



Component diagram



Deployment diagram



Appendix A — Database Schema (generated from the live database)

Generated on 2026-09-27 by querying the project database catalog. This is the authoritative field list.

`alerts`

Column	Type	Nullable	Default
id	uuid	NO	gen_random_uuid()
specialist_id	uuid	NO	
subscriber_id	uuid	NO	
message	text	NO	
is_read	boolean	NO	false
created_at	timestamp with time zone	NO	now()

Constraints

- `alerts_message_check`: CHECK (((char_length(message) >= 2) AND (char_length(message) <= 1000)))
- `alerts_specialist_id_fkey`: FOREIGN KEY (specialist_id) REFERENCES profiles(id) ON DELETE CASCADE
- `alerts_subscriber_id_fkey`: FOREIGN KEY (subscriber_id) REFERENCES profiles(id) ON DELETE CASCADE
- `alerts_pkey`: PRIMARY KEY (id)

Access rules (RLS policies)

- **Delete own alerts** (DELETE, authenticated): USING `((auth.uid() = subscriber_id) OR (auth.uid() = specialist_id))` CHECK –
- **Read own alerts** (SELECT, authenticated): USING `((auth.uid() = subscriber_id) OR (auth.uid() = specialist_id))` CHECK –
- **Specialists send alerts** (INSERT, authenticated): USING – CHECK `((specialist_id = auth.uid()) AND has_role(auth.uid(), 'specialist'::app_role))`
- **Subscriber marks read** (UPDATE, authenticated): USING `(auth.uid() = subscriber_id)` CHECK –

Triggers

`attendance`

Column	Type	Nullable	Default
id	uuid	NO	gen_random_uuid()
subscriber_id	uuid	NO	
group_id	uuid	NO	
attend_date	date	NO	CURRENT_DATE
attend_time	time without time zone	NO	LOCALTIME(0)
exercises	text	NO	
status	text	NO	'recorded'::text
specialist_note	text	YES	
created_at	timestamp with time zone	NO	now()

Constraints

- `attendance_exercises_check`: CHECK (((char_length(exercises) >= 1) AND (char_length(exercises) <= 1000)))
- `attendance_status_check`: CHECK ((status = ANY (ARRAY['recorded'::text, 'reviewed'::text, 'rejected'::text])))

- `attendance_group_id_fkey`: FOREIGN KEY (group_id) REFERENCES subscription_groups(id) ON DELETE SET NULL
- `attendance_subscriber_id_fkey`: FOREIGN KEY (subscriber_id) REFERENCES profiles(id) ON DELETE CASCADE
- `attendance_pkey`: PRIMARY KEY (id)

Access rules (RLS policies)

- **Admins delete attendance** (DELETE, authenticated): USING `has_role(auth.uid(), 'admin'::app_role)` CHECK –
- **Admins update attendance** (UPDATE, authenticated): USING `has_role(auth.uid(), 'admin'::app_role)` CHECK `has_role(auth.uid(), 'admin'::app_role)`
- **Delete own unreviewed attendance** (DELETE, authenticated): USING `((auth.uid() = subscriber_id) AND (status = 'recorded'::text))` CHECK –
- **Read own attendance or staff** (SELECT, authenticated): USING `((auth.uid() = subscriber_id) OR has_role(auth.uid(), 'specialist'::app_role) OR has_role(auth.uid(), 'admin'::app_role))` CHECK –
- **Record own attendance** (INSERT, authenticated): USING – CHECK `((auth.uid() = subscriber_id) AND (status = 'recorded'::text) AND (attend_date = CURRENT_DATE) AND (specialist_note IS NULL) AND is_active_member(auth.uid()))`
- **Specialists review attendance** (UPDATE, authenticated): USING `has_role(auth.uid(), 'specialist'::app_role)` CHECK –

Triggers

- `attendance_validate` BEFORE INSERT → EXECUTE FUNCTION validate_attendance()
- `attendance_validate` BEFORE UPDATE → EXECUTE FUNCTION validate_attendance()

`comments`

Column	Type	Nullable	Default
id	uuid	NO	gen_random_uuid()
post_id	uuid	NO	
author_id	uuid	NO	
content	text	NO	
created_at	timestamp with time zone	NO	now()

Constraints

- `comments_content_check`: CHECK (((char_length(content) >= 1) AND (char_length(content) <= 2000)))
- `comments_author_id_fkey`: FOREIGN KEY (author_id) REFERENCES profiles(id) ON DELETE CASCADE
- `comments_post_id_fkey`: FOREIGN KEY (post_id) REFERENCES posts(id) ON DELETE CASCADE
- `comments_pkey`: PRIMARY KEY (id)

Access rules (RLS policies)

- **Admins delete comments** (DELETE, authenticated): USING `has_role(auth.uid(), 'admin'::app_role)` CHECK –
- **Comments are public** (SELECT, public): USING `true` CHECK –
- **Create own comments** (INSERT, authenticated): USING – CHECK `((auth.uid() = author_id) AND is_active_member(auth.uid()))`
- **Delete own comments** (DELETE, authenticated): USING `(auth.uid() = author_id)` CHECK –

Triggers

`devices`

Column	Type	Nullable	Default
id	uuid	NO	gen_random_uuid()
device_name	text	NO	
guidance	text	NO	

Column	Type	Nullable	Default
instructions	text	YES	
advantages	text	YES	
disadvantages	text	YES	
category	text	NO	'exercise'::text
specialist_id	uuid	NO	
group_id	uuid	YES	
created_at	timestamp with time zone	NO	now()
updated_at	timestamp with time zone	NO	now()
approval_status	text	NO	'pending'::text
rejection_reason	text	YES	

Constraints

- `devices_approval_status_check`: CHECK ((approval_status = ANY (ARRAY['pending'::text, 'approved'::text, 'rejected'::text])))
- `devices_category_check`: CHECK ((category = ANY (ARRAY['exercise'::text, 'nutrition'::text, 'culture'::text])))
- `devices_device_name_check`: CHECK (((char_length(device_name) >= 2) AND (char_length(device_name) <= 150)))
- `devices_group_id_fkey`: FOREIGN KEY (group_id) REFERENCES subscription_groups(id) ON DELETE SET NULL
- `devices_specialist_id_fkey`: FOREIGN KEY (specialist_id) REFERENCES profiles(id) ON DELETE CASCADE
- `devices_pkey`: PRIMARY KEY (id)

Access rules (RLS policies)

- **Admins review guidance** (UPDATE, authenticated): USING `has_role(auth.uid(), 'admin'::app_role)` CHECK `has_role(auth.uid(), 'admin'::app_role)`
- **Approved guidance is public** (SELECT, public): USING `(approval_status = 'approved'::text)` CHECK –
- **Authors and admins see own or all guidance** (SELECT, authenticated): USING `((auth.uid() = specialist_id) OR has_role(auth.uid(), 'admin'::app_role))` CHECK –
- **Owner or admin delete guidance** (DELETE, authenticated): USING `((specialist_id = auth.uid()) OR has_role(auth.uid(), 'admin'::app_role))` CHECK –
- **Specialists add guidance** (INSERT, authenticated): USING – CHECK `((specialist_id = auth.uid()) AND has_role(auth.uid(), 'specialist'::app_role))`
- **Specialists edit own guidance** (UPDATE, authenticated): USING `((specialist_id = auth.uid()) AND has_role(auth.uid(), 'specialist'::app_role))` CHECK –

Triggers

- `devices_approval` BEFORE INSERT → EXECUTE FUNCTION enforce_content_approval()
- `devices_approval` BEFORE UPDATE → EXECUTE FUNCTION enforce_content_approval()
- `devices_updated` BEFORE UPDATE → EXECUTE FUNCTION update_updated_at_column()

`member_contacts`

Column	Type	Nullable	Default
user_id	uuid	NO	
phone	text	YES	
age	integer	YES	
updated_at	timestamp with time zone	NO	now()

Constraints

- `member_contacts_user_id_fkey`: FOREIGN KEY (user_id) REFERENCES profiles(id) ON DELETE CASCADE
- `member_contacts_pkey`: PRIMARY KEY (user_id)

Access rules (RLS policies)

- **Insert own contact** (INSERT, authenticated): USING – CHECK `(auth.uid() = user_id)`
- **Read own contact or admin** (SELECT, authenticated): USING `((auth.uid() = user_id) OR has_role(auth.uid(), 'admin'::app_role))` CHECK –
- **Update own contact** (UPDATE, authenticated): USING `(auth.uid() = user_id)` CHECK `(auth.uid() = user_id)`

Triggers

- `member_contacts_updated` BEFORE UPDATE → EXECUTE FUNCTION update_updated_at_column()

`payment_methods`

Column	Type	Nullable	Default
id	uuid	NO	gen_random_uuid()
name	text	NO	
description	text	YES	
is_active	boolean	NO	true
created_at	timestamp with time zone	NO	now()

Constraints

- `payment_methods_pkey`: PRIMARY KEY (id)

Access rules (RLS policies)

- **Active payment methods are public** (SELECT, public): USING `(is_active OR has_role(auth.uid(), 'admin'::app_role))` CHECK –
- **Admins manage payment methods** (ALL, authenticated): USING `has_role(auth.uid(), 'admin'::app_role)` CHECK `has_role(auth.uid(), 'admin'::app_role)`

Triggers

`payments`

Column	Type	Nullable	Default
id	uuid	NO	gen_random_uuid()
subscriber_id	uuid	NO	
subscription_id	uuid	NO	
payment_method_id	uuid	YES	
value	numeric	NO	
note	text	YES	
status	text	NO	'pending'::text
payment_date	timestamp with time zone	NO	now()

Constraints

- `payments_status_check`: CHECK ((status = ANY (ARRAY['pending'::text, 'confirmed'::text, 'rejected'::text])))
- `payments_value_check`: CHECK ((value >= (0)::numeric))
- `payments_payment_method_id_fkey`: FOREIGN KEY (payment_method_id) REFERENCES payment_methods(id) ON DELETE SET NULL
- `payments_subscriber_id_fkey`: FOREIGN KEY (subscriber_id) REFERENCES profiles(id) ON DELETE CASCADE
- `payments_subscription_id_fkey`: FOREIGN KEY (subscription_id) REFERENCES subscriptions(id) ON DELETE CASCADE
- `payments_pkey`: PRIMARY KEY (id)

Access rules (RLS policies)

- **Admins update payments** (UPDATE, authenticated): USING `has_role(auth.uid(), 'admin'::app_role)` CHECK –
- **Create own payment** (INSERT, authenticated): USING – CHECK `((auth.uid() = subscriber_id) AND (status = 'pending'::text))`
- **Read own payments or admin** (SELECT, authenticated): USING `((auth.uid() = subscriber_id) OR has_role(auth.uid(), 'admin'::app_role))` CHECK –

Triggers

`post_likes`

Column	Type	Nullable	Default
post_id	uuid	NO	
user_id	uuid	NO	
created_at	timestamp with time zone	NO	now()

Constraints

- `post_likes_post_id_fkey`: FOREIGN KEY (post_id) REFERENCES posts(id) ON DELETE CASCADE
- `post_likes_user_id_fkey`: FOREIGN KEY (user_id) REFERENCES profiles(id) ON DELETE CASCADE
- `post_likes_pkey`: PRIMARY KEY (post_id, user_id)

Access rules (RLS policies)

- **Like as self** (INSERT, authenticated): USING – CHECK `(auth.uid() = user_id)`
- **Likes are public** (SELECT, public): USING `true` CHECK –
- **Unlike as self** (DELETE, authenticated): USING `(auth.uid() = user_id)` CHECK –

Triggers

`posts`

Column	Type	Nullable	Default
id	uuid	NO	gen_random_uuid()
author_id	uuid	NO	
post_type	USER-DEFINED	NO	'experience'::post_type
category	USER-DEFINED	NO	'general'::post_category
title	text	NO	
content	text	NO	
created_at	timestamp with time zone	NO	now()
approval_status	text	NO	'pending'::text
rejection_reason	text	YES	

Constraints

- `posts_approval_status_check`: CHECK ((approval_status = ANY (ARRAY['pending'::text, 'approved'::text, 'rejected'::text])))
- `posts_content_check`: CHECK (((char_length(content) >= 1) AND (char_length(content) <= 5000)))
- `posts_title_check`: CHECK (((char_length(title) >= 3) AND (char_length(title) <= 150)))
- `posts_author_id_fkey`: FOREIGN KEY (author_id) REFERENCES profiles(id) ON DELETE CASCADE
- `posts_pkey`: PRIMARY KEY (id)

Access rules (RLS policies)

- **Admins delete posts** (DELETE, authenticated): USING `has_role(auth.uid(), 'admin'::app_role)` CHECK –

- **Admins review posts** (UPDATE, authenticated): USING `has_role(auth.uid(), 'admin'::app_role)` CHECK `has_role(auth.uid(), 'admin'::app_role)`
- **Approved posts are public** (SELECT, public): USING `(approval_status = 'approved'::text)` CHECK –
- **Authors and admins see own or all posts** (SELECT, authenticated): USING `((auth.uid() = author_id) OR has_role(auth.uid(), 'admin'::app_role))` CHECK –
- **Create own posts** (INSERT, authenticated): USING – CHECK `((auth.uid() = author_id) AND is_active_member(auth.uid()))`
- **Delete own posts** (DELETE, authenticated): USING `(auth.uid() = author_id)` CHECK –
- **Update own posts** (UPDATE, authenticated): USING `(auth.uid() = author_id)` CHECK –

Triggers

- `posts_approval` BEFORE INSERT → EXECUTE FUNCTION enforce_content_approval()
- `posts_approval` BEFORE UPDATE → EXECUTE FUNCTION enforce_content_approval()

`profiles`

Column	Type	Nullable	Default
id	uuid	NO	
display_name	text	NO	'Member'::text
bio	text	YES	
fitness_level	text	NO	'beginner'::text
created_at	timestamp with time zone	NO	now()
brief	text	YES	
status	text	NO	'active'::text

Constraints

- `profiles_status_check`: CHECK ((status = ANY (ARRAY['active'::text, 'blocked'::text])))
- `profiles_pkey`: PRIMARY KEY (id)

Access rules (RLS policies)

- **Admins update profiles** (UPDATE, authenticated): USING `has_role(auth.uid(), 'admin'::app_role)` CHECK –
- **Insert own profile** (INSERT, authenticated): USING – CHECK `(auth.uid() = id)`
- **Members read own profile** (SELECT, authenticated): USING `((auth.uid() = id) OR has_role(auth.uid(), 'admin'::app_role))` CHECK –
- **Update own profile** (UPDATE, authenticated): USING `(auth.uid() = id)` CHECK –

Triggers

- `profiles_protect_status` BEFORE UPDATE → EXECUTE FUNCTION protect_profile_status()

`queries`

Column	Type	Nullable	Default
id	uuid	NO	gen_random_uuid()
subscriber_id	uuid	NO	
query_text	text	NO	
query_date	timestamp with time zone	NO	now()
specialist_id	uuid	YES	
answer_text	text	YES	
answer_date	timestamp with time zone	YES	

Constraints

- `queries_query_text_check`: CHECK (((char_length(query_text) >= 3) AND (char_length(query_text) <= 2000)))
- `queries_specialist_id_fkey`: FOREIGN KEY (specialist_id) REFERENCES profiles(id) ON DELETE SET NULL
- `queries_subscriber_id_fkey`: FOREIGN KEY (subscriber_id) REFERENCES profiles(id) ON DELETE CASCADE
- `queries_pkey`: PRIMARY KEY (id)

Access rules (RLS policies)

- **Ask own query** (INSERT, authenticated): USING – CHECK `((auth.uid() = subscriber_id) AND (answer_text IS NULL) AND is_active_member(auth.uid()))`
- **Delete own query** (DELETE, authenticated): USING `(auth.uid() = subscriber_id)` CHECK –
- **Read own queries or staff** (SELECT, authenticated): USING `((auth.uid() = subscriber_id) OR has_role(auth.uid(), 'specialist'::app_role) OR has_role(auth.uid(), 'admin'::app_role))` CHECK –
- **Specialists answer** (UPDATE, authenticated): USING `has_role(auth.uid(), 'specialist'::app_role)` CHECK `(specialist_id = auth.uid())`

Triggers

`subscription_groups`

Column	Type	Nullable	Default
id	uuid	NO	gen_random_uuid()
title	text	NO	
description	text	YES	
features	ARRAY	NO	'{}'::text[]
price	numeric	NO	0
begin_date	date	YES	
end_date	date	YES	
specialist_id	uuid	YES	
is_active	boolean	NO	true
created_at	timestamp with time zone	NO	now()
updated_at	timestamp with time zone	NO	now()
capacity	integer	YES	
subscriber_count	integer	NO	0

Constraints

- `subscription_groups_capacity_check`: CHECK (((capacity IS NULL) OR (capacity > 0)))
- `subscription_groups_price_check`: CHECK ((price >= (0)::numeric))
- `subscription_groups_specialist_id_fkey`: FOREIGN KEY (specialist_id) REFERENCES profiles(id) ON DELETE SET NULL
- `subscription_groups_pkey`: PRIMARY KEY (id)

Access rules (RLS policies)

- **Active groups are public** (SELECT, public): USING `(is_active OR has_role(auth.uid(), 'admin'::app_role))` CHECK –
- **Admins manage groups** (ALL, authenticated): USING `has_role(auth.uid(), 'admin'::app_role)` CHECK `has_role(auth.uid(), 'admin'::app_role)`

Triggers

- `groups_updated` BEFORE UPDATE → EXECUTE FUNCTION update_updated_at_column()

`subscriptions`

Column	Type	Nullable	Default
id	uuid	NO	gen_random_uuid()
subscriber_id	uuid	NO	
group_id	uuid	NO	
status	text	NO	'pending'::text
created_at	timestamp with time zone	NO	now()
updated_at	timestamp with time zone	NO	now()

Constraints

- `subscriptions_status_check`: CHECK ((status = ANY (ARRAY['pending'::text, 'active'::text, 'rejected'::text])))
- `subscriptions_group_id_fkey`: FOREIGN KEY (group_id) REFERENCES subscription_groups(id) ON DELETE CASCADE
- `subscriptions_subscriber_id_fkey`: FOREIGN KEY (subscriber_id) REFERENCES profiles(id) ON DELETE CASCADE
- `subscriptions_pkey`: PRIMARY KEY (id)

Access rules (RLS policies)

- **Admins delete subscriptions** (DELETE, authenticated): USING `has_role(auth.uid(), 'admin'::app_role)` CHECK –
- **Admins update subscriptions** (UPDATE, authenticated): USING `has_role(auth.uid(), 'admin'::app_role)` CHECK –
- **Read own or staff** (SELECT, authenticated): USING `((auth.uid() = subscriber_id) OR has_role(auth.uid(), 'admin'::app_role) OR has_role(auth.uid(), 'specialist'::app_role))` CHECK –
- **Subscribe self** (INSERT, authenticated): USING – CHECK `((auth.uid() = subscriber_id) AND (status = 'pending'::text) AND is_active_member(auth.uid()))`

Triggers

- `subs_updated` BEFORE UPDATE → EXECUTE FUNCTION update_updated_at_column()
- `subscriptions_check_space` BEFORE INSERT → EXECUTE FUNCTION check_group_space()
- `subscriptions_check_space` BEFORE UPDATE → EXECUTE FUNCTION check_group_space()
- `subscriptions_sync_count` AFTER DELETE → EXECUTE FUNCTION after_subscription_change()
- `subscriptions_sync_count` AFTER INSERT → EXECUTE FUNCTION after_subscription_change()
- `subscriptions_sync_count` AFTER UPDATE → EXECUTE FUNCTION after_subscription_change()

`user_roles`

Column	Type	Nullable	Default
id	uuid	NO	gen_random_uuid()
user_id	uuid	NO	
role	USER-DEFINED	NO	
created_at	timestamp with time zone	NO	now()

Constraints

- `user_roles_user_id_fkey`: FOREIGN KEY (user_id) REFERENCES auth.users(id) ON DELETE CASCADE
- `user_roles_pkey`: PRIMARY KEY (id)
- `user_roles_user_id_role_key`: UNIQUE (user_id, role)

Access rules (RLS policies)

- **Admins manage roles** (ALL, authenticated): USING `has_role(auth.uid(), 'admin'::app_role)` CHECK `has_role(auth.uid(), 'admin'::app_role)`
- **Read own roles or admin** (SELECT, authenticated): USING `((auth.uid() = user_id) OR has_role(auth.uid(), 'admin'::app_role))` CHECK –

Triggers

Enumerated types

- `aal_level`: aal1, aal2, aal3
- `action`: INSERT, UPDATE, DELETE, TRUNCATE, ERROR
- `app_role`: admin, specialist, subscriber
- `buckettype`: STANDARD, ANALYTICS, VECTOR
- `code_challenge_method`: s256, plain
- `equality_op`: eq, neq, lt, lte, gt, gte, in, like, ilike, is, match, imatch, isdistinct
- `factor_status`: unverified, verified
- `factor_type`: totp, webauthn, phone, recovery_code
- `oauth_authorization_status`: pending, approved, denied, expired
- `oauth_client_type`: public, confidential
- `oauth_registration_type`: dynamic, manual
- `oauth_response_type`: code
- `one_time_token_type`: confirmation_token, reauthentication_token, recovery_token, email_change_token_new, email_change_token_current, phone_change_token
- `post_category`: strength, cardio, nutrition, flexibility, recovery, general
- `post_type`: challenge, solution, experience, tip

Indexes

- `alerts_pkey` on alerts: `CREATE UNIQUE INDEX alerts_pkey ON public.alerts USING btree (id)`
- `attendance_pkey` on attendance: `CREATE UNIQUE INDEX attendance_pkey ON public.attendance USING btree (id)`
- `attendance_one_per_day` on attendance: `CREATE UNIQUE INDEX attendance_one_per_day ON public.attendance USING btree (subscriber_id, group_id, attend_date)`
- `comments_pkey` on comments: `CREATE UNIQUE INDEX comments_pkey ON public.comments USING btree (id)`
- `devices_pkey` on devices: `CREATE UNIQUE INDEX devices_pkey ON public.devices USING btree (id)`
- `member_contacts_pkey` on member_contacts: `CREATE UNIQUE INDEX member_contacts_pkey ON public.member_contacts USING btree (user_id)`
- `payment_methods_pkey` on payment_methods: `CREATE UNIQUE INDEX payment_methods_pkey ON public.payment_methods USING btree (id)`
- `payments_pkey` on payments: `CREATE UNIQUE INDEX payments_pkey ON public.payments USING btree (id)`
- `post_likes_pkey` on post_likes: `CREATE UNIQUE INDEX post_likes_pkey ON public.post_likes USING btree (post_id, user_id)`
- `posts_pkey` on posts: `CREATE UNIQUE INDEX posts_pkey ON public.posts USING btree (id)`
- `profiles_pkey` on profiles: `CREATE UNIQUE INDEX profiles_pkey ON public.profiles USING btree (id)`
- `queries_pkey` on queries: `CREATE UNIQUE INDEX queries_pkey ON public.queries USING btree (id)`
- `subscription_groups_pkey` on subscription_groups: `CREATE UNIQUE INDEX subscription_groups_pkey ON public.subscription_groups USING btree (id)`
- `subscriptions_pkey` on subscriptions: `CREATE UNIQUE INDEX subscriptions_pkey ON public.subscriptions USING btree (id)`
- `subscriptions_one_per_group` on subscriptions: `CREATE UNIQUE INDEX subscriptions_one_per_group ON public.subscriptions USING btree (subscriber_id, group_id) WHERE (status <> 'rejected'::text)`
- `user_roles_user_id_role_key` on user_roles: `CREATE UNIQUE INDEX user_roles_user_id_role_key ON public.user_roles USING btree (user_id, role)`
- `user_roles_pkey` on user_roles: `CREATE UNIQUE INDEX user_roles_pkey ON public.user_roles USING btree (id)`

Database functions (full bodies)

after_subscription_change

```
CREATE OR REPLACE FUNCTION public.after_subscription_change()
 RETURNS trigger
 LANGUAGE plpgsql
 SECURITY DEFINER
```

```
 SET search_path TO 'public'
AS $function$
BEGIN
  IF TG_OP IN ('INSERT','UPDATE') THEN PERFORM public.sync_group_count(NEW.group_id); END IF;
  IF TG_OP IN ('DELETE','UPDATE') THEN PERFORM public.sync_group_count(OLD.group_id); END IF;
  RETURN NULL;
END; $function$
```

check_group_space

```
CREATE OR REPLACE FUNCTION public.check_group_space()
 RETURNS trigger
 LANGUAGE plpgsql
 SECURITY DEFINER
 SET search_path TO 'public'
AS $function$
DECLARE cap int; cnt int;
BEGIN
  IF NEW.status = 'rejected' THEN RETURN NEW; END IF;
  IF TG_OP = 'UPDATE' AND OLD.status <> 'rejected' AND OLD.group_id = NEW.group_id THEN RETURN NEW; END IF;
  SELECT capacity INTO cap FROM public.subscription_groups WHERE id = NEW.group_id FOR UPDATE;
  IF EXISTS (SELECT 1 FROM public.subscriptions WHERE subscriber_id = NEW.subscriber_id AND group_id =
        NEW.group_id AND status <> 'rejected' AND id <> NEW.id) THEN
    RAISE EXCEPTION 'ALREADY_REGISTERED';
  END IF;
  IF cap IS NOT NULL THEN
    SELECT count(*) INTO cnt FROM public.subscriptions WHERE group_id = NEW.group_id AND status <>
        'rejected' AND id <> NEW.id;
    IF cnt >= cap THEN RAISE EXCEPTION 'NO_SPACE_IN_GROUP'; END IF;
  END IF;
  RETURN NEW;
END; $function$
```

enforce_content_approval

```
CREATE OR REPLACE FUNCTION public.enforce_content_approval()
 RETURNS trigger
 LANGUAGE plpgsql
 SECURITY DEFINER
 SET search_path TO 'public'
AS $function$
BEGIN
  IF public.has_role(auth.uid(), 'admin') THEN
    IF TG_OP = 'INSERT' THEN NEW.approval_status := 'approved'; NEW.rejection_reason := NULL; END IF;
    IF NEW.approval_status <> 'rejected' THEN NEW.rejection_reason := NULL; END IF;
    RETURN NEW;
  END IF;
  IF TG_OP = 'INSERT' THEN
    NEW.approval_status := 'pending'; NEW.rejection_reason := NULL;
  ELSIF NEW.approval_status IS DISTINCT FROM OLD.approval_status
     OR NEW.rejection_reason IS DISTINCT FROM OLD.rejection_reason
     OR to_jsonb(NEW) - 'approval_status' - 'rejection_reason' - 'updated_at' IS DISTINCT FROM to_jsonb(OLD)
        - 'approval_status' - 'rejection_reason' - 'updated_at' THEN
    NEW.approval_status := 'pending'; NEW.rejection_reason := NULL;
  END IF;
  RETURN NEW;
END; $function$
```

handle_new_user

```
CREATE OR REPLACE FUNCTION public.handle_new_user()
 RETURNS trigger
 LANGUAGE plpgsql
 SECURITY DEFINER
 SET search_path TO 'public'
```

```
AS $function$
BEGIN
  INSERT INTO public.profiles (id, display_name)
  VALUES (NEW.id, COALESCE(NEW.raw_user_meta_data->>'display_name', NEW.raw_user_meta_data->>'full_name',
        split_part(NEW.email,'@',1), 'Member'));
  INSERT INTO public.member_contacts (user_id, phone, age)
  VALUES (NEW.id, NULLIF(NEW.raw_user_meta_data->>'phone',''), NULLIF(NEW.raw_user_meta_data-
        >>'age','')::int);
  INSERT INTO public.user_roles (user_id, role) VALUES (NEW.id, 'subscriber');
  IF NOT EXISTS (SELECT 1 FROM public.user_roles WHERE role = 'admin') THEN
    INSERT INTO public.user_roles (user_id, role) VALUES (NEW.id, 'admin');
  END IF;
  RETURN NEW;
END; $function$
```

has_role

```
CREATE OR REPLACE FUNCTION public.has_role(_user_id uuid, _role app_role)
 RETURNS boolean
 LANGUAGE sql
 STABLE SECURITY DEFINER
 SET search_path TO 'public'
AS $function$
  SELECT EXISTS (SELECT 1 FROM public.user_roles WHERE user_id = _user_id AND role = _role)
$function$
```

is_active_member

```
CREATE OR REPLACE FUNCTION public.is_active_member(_user_id uuid)
 RETURNS boolean
 LANGUAGE sql
 STABLE SECURITY DEFINER
 SET search_path TO 'public'
AS $function$
  SELECT EXISTS (SELECT 1 FROM public.profiles WHERE id = _user_id AND status = 'active')
$function$
```

protect_profile_status

```
CREATE OR REPLACE FUNCTION public.protect_profile_status()
 RETURNS trigger
 LANGUAGE plpgsql
 SECURITY DEFINER
 SET search_path TO 'public'
AS $function$
BEGIN
  IF NEW.status IS DISTINCT FROM OLD.status AND NOT public.has_role(auth.uid(),'admin') THEN
    NEW.status := OLD.status;
  END IF;
  RETURN NEW;
END; $function$
```

sync_group_count

```
CREATE OR REPLACE FUNCTION public.sync_group_count(_gid uuid)
 RETURNS void
 LANGUAGE sql
 SECURITY DEFINER
 SET search_path TO 'public'
AS $function$
  UPDATE public.subscription_groups SET subscriber_count =
```

```
    (SELECT count(*) FROM public.subscriptions WHERE group_id = _gid AND status <> 'rejected') WHERE id =
        _gid;
$function$
```

update_updated_at_column

```
CREATE OR REPLACE FUNCTION public.update_updated_at_column()
 RETURNS trigger
 LANGUAGE plpgsql
 SET search_path TO 'public'
AS $function$ BEGIN NEW.updated_at = now(); RETURN NEW; END; $function$
```

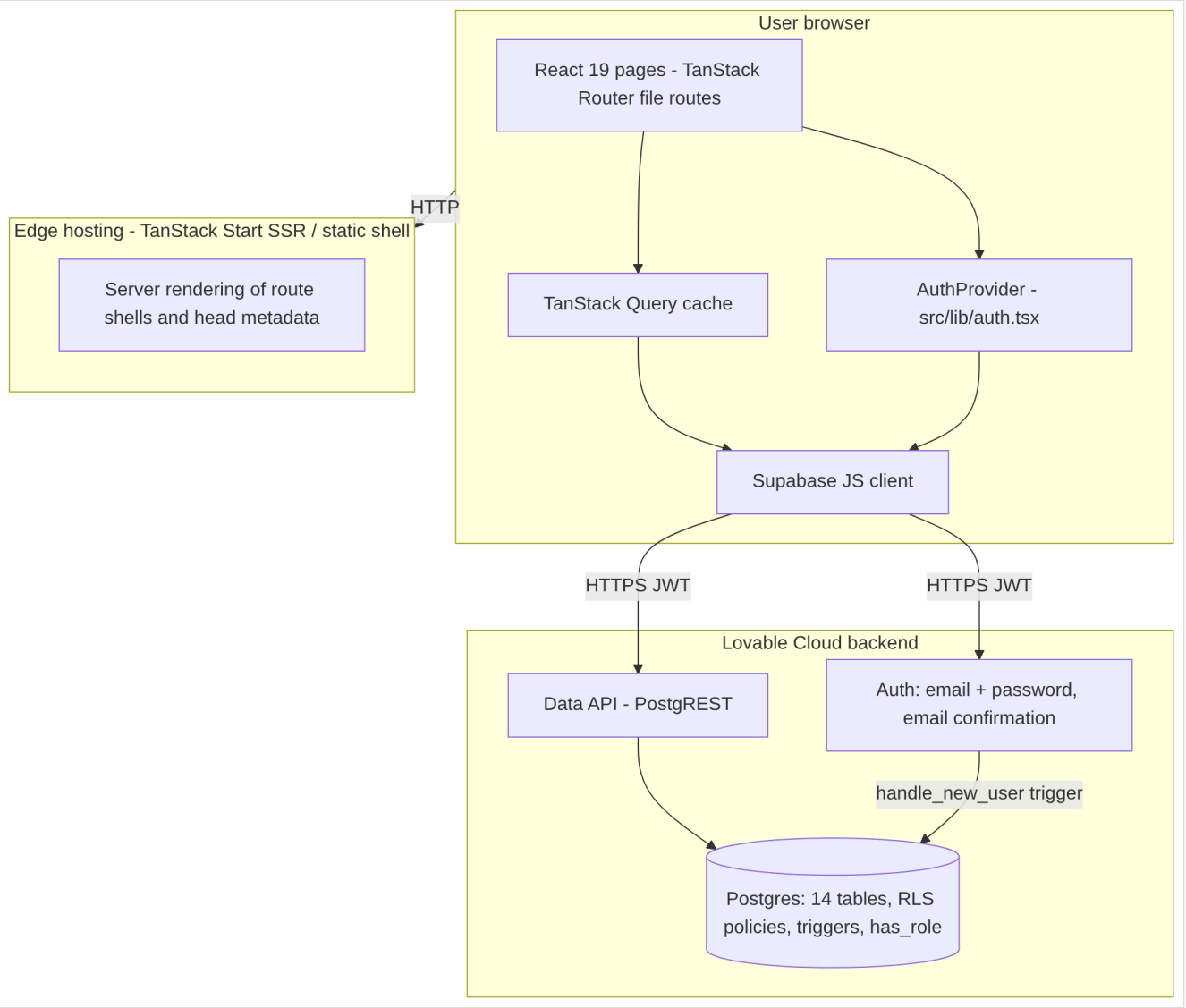
validate_attendance

```
CREATE OR REPLACE FUNCTION public.validate_attendance()
 RETURNS trigger
 LANGUAGE plpgsql
 SECURITY DEFINER
 SET search_path TO 'public'
AS $function$
DECLARE b date; e date;
BEGIN
  IF TG_OP = 'UPDATE' AND NEW.subscriber_id = OLD.subscriber_id AND NEW.group_id = OLD.group_id AND
       NEW.attend_date = OLD.attend_date THEN RETURN NEW; END IF;
  IF NOT EXISTS (SELECT 1 FROM public.subscriptions WHERE subscriber_id = NEW.subscriber_id AND group_id =
       NEW.group_id AND status = 'active') THEN
    RAISE EXCEPTION 'NOT_ACTIVE_IN_GROUP';
  END IF;
  SELECT begin_date, end_date INTO b, e FROM public.subscription_groups WHERE id = NEW.group_id;
  IF NEW.attend_date > CURRENT_DATE OR (b IS NOT NULL AND NEW.attend_date < b) OR (e IS NOT NULL AND
       NEW.attend_date > e) THEN
    RAISE EXCEPTION 'OUTSIDE_GROUP_PERIOD';
  END IF;
  IF EXISTS (SELECT 1 FROM public.attendance WHERE subscriber_id = NEW.subscriber_id AND group_id =
       NEW.group_id AND attend_date = NEW.attend_date AND id <> NEW.id) THEN
    RAISE EXCEPTION 'ATTENDANCE_ALREADY_RECORDED';
  END IF;
  RETURN NEW;
END; $function$
```

Migration history

- supabase/migrations/20260926203953_1fc87fb2-4037-411b-b7a9-322fd28e3401.sql
- supabase/migrations/20260926204003_f7845333-23b4-4325-9616-dd7df73e1110.sql
- supabase/migrations/20260926211158_cf5aeef8-57e0-4864-b695-5d4497fdc319.sql
- supabase/migrations/20260926211212_fd61bb01-2dc7-40a6-b2a1-797e9afa938b.sql
- supabase/migrations/20260926212134_b6465077-eaca-4483-92f7-e5e68a5c544b.sql
- supabase/migrations/20260926213503_d62c7fab-914e-46a6-9878-8c8794c20692.sql
- supabase/migrations/20260926213511_540b1e18-061d-4749-9d9a-4140a7126044.sql
- supabase/migrations/20260926213935_376ce150-f037-43b3-adef-6b8e04f78331.sql
- supabase/migrations/20260926215502_babe47c2-aaca-4f35-91cc-d1c9ff0e1169.sql
- supabase/migrations/20260926215747_3b567569-841c-4e14-9cda-54cc5c3c9e04.sql

10 — System Architecture



Frontend

- React 19 single-page experience with server-rendered shells via TanStack Start.
- File-based routes in `src/routes/` (one file per page; `__root.tsx` is the shared layout with header, footer, `<Outlet/>`, toaster, auth and query providers).
- TanStack Query caches reads (`useQuery` with keys such as `groups`, `my-alerts`, `admin-subs`) and is invalidated after writes.
- UI primitives: shadcn/ui (Radix) in `src/components/ui/`; project components `SiteHeader`, `SiteFooter`, `PostCard`.

Application structure

```
Browser page (route component)
  ├── useAuth()  → session, roles, blocked       (src/lib/auth.tsx)
  ├── helpers    → fetchGroups, fetchPosts, ...  (src/lib/fitness.ts, community.ts)
  └── supabase client (src/integrations/supabase/client.ts)
          │ HTTPS + user JWT
          ▼
Lovable Cloud: Auth  +  Data API  →  Postgres (RLS policies + triggers)
```

Database

Postgres with RLS on all tables; business rules in triggers (document 08).

Authentication

Email/password with email confirmation. `AuthProvider` subscribes to `onAuthStateChange`, stores the session, and loads roles from `user_roles` and block status from `profiles`.

Data flow

1. Component calls the Supabase client (select/insert/update/delete).
2. The request carries the user's token; the database evaluates policies with `auth.uid()` and `has_role()`.
3. Triggers validate or adjust the row (e.g. set `pending`, raise `NO_SPACE_IN_GROUP`).
4. The component shows a toast (errors mapped to Arabic by `subscribeErrorMsg` / `attendanceErrorMsg`) and invalidates cached queries.

Server-side code

There are **no custom server functions or API routes**. `src/server.ts` and `src/start.ts` are framework bootstrap and error-capture only. `src/integrations/supabase/auth-middleware.ts`, `auth-attacher.ts`, `client.server.ts`, `cron-auth.ts` are auto-generated platform files not used by the app's pages.

External services

Only Lovable Cloud (auth + database) and Google Fonts (Cairo, Tajawal). No payment gateway, AI, email provider configuration or third-party APIs are used by the app code.

11 — Source Code Documentation

Complete source of every hand-written file is in **Appendix B**. This document explains structure and the most important code.

Project file structure

```
.
├── AGENTS.md                       Architecture decisions (one rule + reason each)
├── package.json                    Dependencies and scripts (vite dev/build)
├── vite.config.ts                  Vite + TanStack Start config
├── public/
│   ├── favicon.png                 Custom My Fitness icon
│   └── robots.txt
├── supabase/
│   ├── config.toml                 Backend project config (auto-managed)
│   └── migrations/*.sql            10 migrations: schema, policies, triggers, seeds
├── docs/                           This documentation package
└── src/
    ├── router.tsx                  Creates the router + QueryClient context
    ├── routeTree.gen.ts            Auto-generated route tree (do not edit)
    ├── start.ts / server.ts        Framework bootstrap, error middleware
    ├── styles.css                  Tailwind v4 theme tokens, fonts, colors
    ├── components/
    │   ├── SiteHeader.tsx          Header nav (7 links), auth buttons, SiteFooter
    │   ├── PostCard.tsx            Post summary card + TypeBadge
    │   └── ui/                     shadcn/ui primitives (button, card, tabs, ...)
    ├── hooks/use-mobile.tsx        Viewport helper (shadcn)
    ├── integrations/supabase/      Auto-generated backend client + types
    ├── lib/
    │   ├── auth.tsx                AuthProvider, useAuth, roles, ROLE_LABEL
    │   ├── community.ts            Post types/categories, fetchPosts, APPROVAL_LABEL
    │   ├── fitness.ts              Groups, payment methods, capacity + error messages
    │   ├── utils.ts                cn() class merge helper
    │   └── error-*.ts              Error capture/reporting (platform)
    └── routes/
        ├── __root.tsx              HTML shell (lang=ar dir=rtl), providers, layout
        ├── index.tsx               Home
        ├── auth.tsx                Registration + login
        ├── verify.tsx              Email verification
        ├── plans.tsx               Packages list
        ├── checkout.$id.tsx        Subscription + payment request
        ├── dashboard.tsx           Subscriber + trainer dashboard
        ├── admin.tsx               Admin panel (6 tabs)
        ├── exercises.tsx           Trainer guidance (list + form)
        ├── health.tsx              Health article, BMI & water calculator
        ├── test.tsx                Self-assessment quizzes
        ├── community.tsx           Community list with filters
        ├── posts.$id.tsx           Post detail, comments, likes
        ├── new.tsx                 New post form
        ├── profile.tsx             Profile editing
        └── about.tsx               In-app academic documentation
```

Key modules

`src/lib/auth.tsx` — authentication state and roles

Provides `AuthProvider` and `useAuth()` returning `user`, `session`, `loading`, `roles`, `isAdmin`, `isSpecialist`, `blocked`, `signOut`.

```
const loadRoles = useCallback(async (uid: string | undefined) => {
  if (!uid) { setRoles([]); setBlocked(false); return; }
  const [{ data: r }, { data: p }] = await Promise.all([
```

```
    supabase.from("user_roles").select("role").eq("user_id", uid),
    supabase.from("profiles").select("status").eq("id", uid).maybeSingle(),
  ]);
  setRoles((r ?? []).map((x) => x.role as AppRole));
  setBlocked(p?.status === "blocked");
}, []);
```

`onAuthStateChange` updates the session and reloads roles; roles only drive the UI.

`src/lib/fitness.ts` — groups, capacity and error mapping

```
export const NO_SPACE_MSG = "لا يوجد مكان في المجموعة";
export const ALREADY_MSG = "أنت مسجّل بالفعل في هذه المجموعة";
export const isFull = (g: Pick<Group, "capacity" | "subscriber_count">) =>
  g.capacity != null && g.subscriber_count >= g.capacity;
export function subscribeErrorMsg(message: string) {
  if (message.includes("NO_SPACE_IN_GROUP")) return NO_SPACE_MSG;
  if (message.includes("ALREADY_REGISTERED") || message.includes("subscriptions_one_per_group")) return
      ALREADY_MSG;
  return message;
}
```

Also `fetchGroups(activeOnly)`, `fetchPaymentMethods(activeOnly)`, `STATUS_LABEL`, `GUIDE_CATEGORY`, `sar()`, `attendanceErrorMsg()`.

`src/lib/community.ts` — posts

`fetchPosts(filters)` shows only approved posts unless filtering by author (then access rules decide):

```
if (filters.author) query = query.eq("author_id", filters.author);
else query = query.eq("approval_status", "approved");
```

`src/routes/checkout.$id.tsx` — subscription request

```
const [{ data: fresh }, { data: existing }] = await Promise.all([
  supabase.from("subscription_groups").select("capacity,subscriber_count").eq("id", group.id).maybeSingle(),
  supabase.from("subscriptions").select("id").eq("subscriber_id", user.id).eq("group_id",
group.id).neq("status", "rejected").limit(1),
]);
if (existing?.length) { setBusy(false); return void toast.error(ALREADY_MSG); }
if (fresh && isFull(fresh)) { ...; return void toast.error(NO_SPACE_MSG); }
const { data: sub, error } = await supabase.from("subscriptions")
  .insert({ subscriber_id: user.id, group_id: group.id }).select("id").single();
if (error) { ...; return void toast.error(subscribeErrorMsg(error.message)); }
await supabase.from("payments").insert({ subscriber_id: user.id, subscription_id: sub.id,
  payment_method_id: method, value: group.price, note: note.trim().slice(0, 500) || null });
```

`src/routes/dashboard.tsx` — attendance recording (`MyAttendance`)

```
if (blocked) return void toast.error("حسابك محظور");
if (!selected) return void toast.error("اختر المجموعة");
if (doneToday) return void toast.error(attendanceErrorMsg("ATTENDANCE_ALREADY_RECORDED"));
if (!ex.trim()) return void toast.error("حدّد التمارين التي أديتها");
const { error } = await supabase.from("attendance")
  .insert({ subscriber_id: user!.id, group_id: selected, exercises: ex.trim().slice(0, 1000) });
if (error) return void toast.error(attendanceErrorMsg(error.message));
```

Other components in the file: `MySubscriptions`, `MyQueries`, `MyAlerts`, `MyContent`, `ReviewAttendance`, `AnswerQueries`, alert sender, `Dashboard` (tabs).

`src/routes/exercises.tsx` — guidance submission (`GuideForm`)

```
const { error } = edit
  ? await supabase.from("devices").update(row).eq("id", edit.id)
  : await supabase.from("devices").insert({ ...row, specialist_id: user!.id });
if (error) return void toast.error(error.message);
toast.success("تم إرسال الإرشاد للمشرف، وسيُنشر بعد موافقته");
```

`src/routes/admin.tsx` — content decision (`ContentReview`)

```
const decide = async (table: "devices" | "posts", id: string, ok: boolean) => {
  let reason: string | null = null;
  if (!ok) {
    reason = prompt("سبب الرفض (يظهر للكاتب فقط):")?.trim().slice(0, 500) ?? null;
    if (!reason) return void toast.error("أدخل سبب الرفض");
  }
  const { error } = await supabase.from(table)
    .update({ approval_status: ok ? "approved" : "rejected", rejection_reason: reason }).eq("id", id);
  ...
};
```

Capacity update guard (`Groups.setCap`):

```
if (capacity !== null && !(capacity > 0)) return void toast.error("السعة يجب أن تكون رقمًا أكبر من صفر");
if (capacity !== null && capacity < cur) return void toast.error(`لا يمكن أن تكون السعة أقل من عدد المشتركين الحاليين
($${cur})`);
```

Other components: `Users`, `Requests`, `Groups`, `Methods`, `AttendanceAdmin`, `Admin` (tabs, admin-only gate).

`src/routes/__root.tsx` — shell and layout

```
<html lang="ar" dir="rtl"> ...
<QueryClientProvider client={queryClient}>
  <AuthProvider>
    <SiteHeader /><main className="min-h-[70vh]"><Outlet /></main><SiteFooter /><Toaster />
  </AuthProvider>
</QueryClientProvider>
```

Database logic (real SQL)

```
-- enforce_content_approval() (posts, devices)
IF public.has_role(auth.uid(), 'admin') THEN
  IF TG_OP = 'INSERT' THEN NEW.approval_status := 'approved'; NEW.rejection_reason := NULL; END IF;
  IF NEW.approval_status <> 'rejected' THEN NEW.rejection_reason := NULL; END IF;
  RETURN NEW;
END IF;
IF TG_OP = 'INSERT' THEN NEW.approval_status := 'pending'; NEW.rejection_reason := NULL;
ELSIF <status, reason or any content column changed> THEN
  NEW.approval_status := 'pending'; NEW.rejection_reason := NULL;
END IF;
```

```
-- check_group_space() (subscriptions)
SELECT capacity INTO cap FROM public.subscription_groups WHERE id = NEW.group_id FOR UPDATE;
IF EXISTS (... same subscriber, same group, status <> 'rejected' ...) THEN RAISE EXCEPTION
        'ALREADY_REGISTERED'; END IF;
IF cap IS NOT NULL THEN
  SELECT count(*) INTO cnt FROM public.subscriptions WHERE group_id = NEW.group_id AND status <> 'rejected'
       AND id <> NEW.id;
  IF cnt >= cap THEN RAISE EXCEPTION 'NO_SPACE_IN_GROUP'; END IF;
END IF;
```

```
-- validate_attendance() (attendance)
IF NOT EXISTS (active subscription for subscriber in group) THEN RAISE EXCEPTION 'NOT_ACTIVE_IN_GROUP'; END
        IF;
```

```
IF NEW.attend_date > CURRENT_DATE OR before begin_date OR after end_date THEN RAISE EXCEPTION
        'OUTSIDE_GROUP_PERIOD'; END IF;
IF EXISTS (same subscriber, group, date) THEN RAISE EXCEPTION 'ATTENDANCE_ALREADY_RECORDED'; END IF;
```

(The `<...>` parts summarise conditions; exact bodies are in Appendix A.)

Validation logic summary

Where	Rule
`auth.tsx` Zod	name 2–50, email, password ≥ 6, phone pattern, age 10–100
`new.tsx` Zod	title 3–150, content required
forms	`maxLength` and `.slice()` limits on all text inputs
database CHECKs	title/content/comment/message/exercise lengths, price ≥ 0, statuses, age 5–120
triggers	capacity, duplicates, attendance validity, approval

Role-based access control in code

- UI: `useAuth().isAdmin / isSpecialist / blocked` to show tabs, buttons and gates.
- Database: `has_role()` and `auth.uid()` in 54 policies (document 13).

State management

Local `useState` for forms; TanStack Query for server data; React context (`AuthProvider`) for session. No global store library.

12 — UI / UX Documentation

Main layout

Sticky header (logo "لياقتي" with a square "ل" mark, 7 navigation links, auth buttons) → page content (`max-w-6xl` centred) → footer with academic and non-medical disclaimer. Defined in `__root.tsx` and `SiteHeader.tsx`.

Navigation

Header links: الرئيسية (Home), الرياضة والغذاء (Sport and food), الصحة (Health), اختبر نفسك (Test yourself), الباقات (Packages), المجتمع (Community), التوثيق (Documentation). Signed out: دخول (Login), تسجيل (Register). Signed in: الإدارة (Admin, admins only), لوحتي (My dashboard), ملفي (Profile), خروج (Log out). Active link is highlighted.

RTL / Arabic

`<html lang="ar" dir="rtl">`; logical Tailwind utilities (`ms-auto`, `me-*`); Cairo for headings and Tajawal for body text; email fields use `dir="ltr"`.

Visual design

Warm off-white background, near-black "ink" text, lime-green primary buttons, teal accent; semantic colour tokens in `src/styles.css`. Custom green/teal fitness favicon.

Forms

Labelled inputs (shadcn `Input`, `Textarea`, `Label`), radio lists for payment methods, native selects for categories/groups. Submit buttons disable while busy.

Buttons, cards, tables

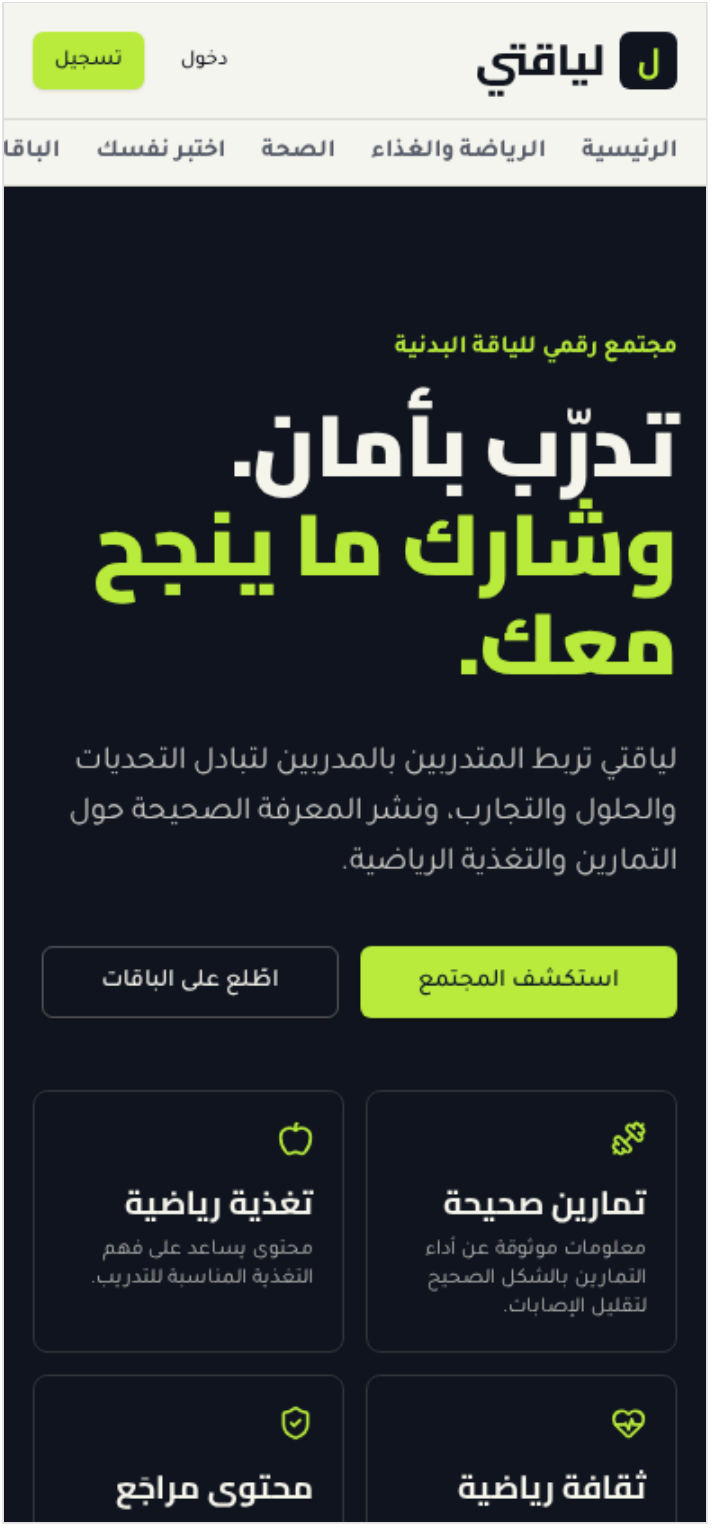
- Buttons: primary, outline, secondary, ghost, destructive variants.
- Cards: `PostCard`, package cards (featured card ringed), guidance cards, dashboard `Card` panels.
- Lists instead of HTML tables in admin/dashboard (each row a bordered block with actions).
- Badges for post type and approval status.

Dashboards

Tabbed (`Tabs`): dashboard (Subscriber / Trainer), admin (6 tabs).

Responsive behaviour

Below the `lg` breakpoint the navigation moves to a horizontally scrollable bar under the header; grids collapse to one column; "ملفي" hides on very small screens.


لياقتي
دخول
تسجيل
الرئيسية
الرياضة والغذاء
الصحة
اختبر نفسك
الباقا
مجتمع رقمي للياقة البدنية
تدرّب بأمان.
وشارك ما ينجح معك.
لياقتي تربط المتدربين بالمدربين لتبادل التحديات والحلول والتجارب، ونشر المعرفة الصحيحة حول التمارين والتغذية الرياضية.
استكشف المجتمع
اطّلع على الباقات
تمارين صحيحة
معلومات موثوقة عن أداء التمارين بالشكل الصحيح لتقليل الإصابات.
تغذية رياضية
محتوى يساعد على فهم التغذية المناسبة للتدريب.
ثقافة رياضية
محتوى مراجع



Validation and error states

- Toasts (sonner) for success and errors in Arabic.
- Inline alert "لا يوجد مكان في المجموعة" with `role="alert"` on full groups.
- Empty state "لا يوجد إرشاد متاح" on guidance.
- "يرجى تسجيل الدخول" gates on protected pages; "هذه الصفحة للمشرف فقط" on admin.
- Loading text "جارٍ التحميل…".
- 404 page and error boundary with retry.

Screenshots

See document 04 and `screenshots/`.

13 — Security

Only mechanisms verified in the source code or database catalog are listed.

Authentication

- Email + password accounts managed by Lovable Cloud Auth; email confirmation required before login (`/verify`, resend supported).
- Password storage and session tokens are handled by the managed auth service; the app never stores passwords itself. (Hashing details belong to the managed service and were not inspected.)
- No social login, no anonymous sign-up.

Authorization and RBAC

- Roles in a separate `user_roles` table; checked by the `SECURITY DEFINER` function `has_role()` to avoid recursive policies and privilege escalation.
- Only admins can insert/delete roles (policy "Admins manage roles"); users can read only their own roles.
- First registered user becomes admin automatically; no hard-coded credentials, no client-side role storage.

Row-level security (54 policies, all tables)

Highlights (full list in Appendix A):

- `profiles`: readable by the owner or an admin only; members update own; admins update any; `protect_profile_status` blocks self-unblocking.
- `member_contacts`: phone/age readable by owner or admin only.
- `posts`, `devices`: public sees `approved` only; authors/admins see own/all; only admins can change approval.
- `subscriptions`, `payments`: subscribers insert their own pending rows; only admins change status.
- `attendance`: insert only own, today, `recorded`, active member; specialists/admins review.
- `queries`, `alerts`: owner/specialist scoped.
- Blocked members: `is_active_member()` in insert policies for posts, comments, subscriptions, queries, attendance.

Validation

Client: Zod schemas and length limits. Server: CHECK constraints and validation triggers (document 08) that cannot be bypassed from the browser.

Access restrictions in the UI

Protected pages render sign-in prompts; admin page requires `isAdmin`. These are convenience only; the database enforces access.

Security scan status

Earlier security findings on overly permissive read rules were fixed (profiles, roles, contacts, groups, payment methods). Two older platform warnings about signed-in users being able to execute built-in helper functions were reported and left unchanged.

Not implemented / not claimed

No custom encryption, rate limiting, CAPTCHA, audit log, two-factor authentication, or payment security (no payments are processed).

14 — Testing

Test approach and evidence

- **Automated page checks (27 Sep 2026):** Playwright opened every public page and captured screenshots (`screenshots/`).
- **Backend probes (27 Sep 2026):** requests to the Data API with the public (visitor) key.
- **Database rule tests (during development):** transactional SQL tests that were rolled back, recorded in the project history for capacity, duplicate registration, rejection/deletion freeing seats, attendance rules and hidden rejected content.
- **Type check:** the project compiles (build log "build OK").
- **Not executed:** signed-in end-to-end UI tests — the database has no user accounts, so no session could be created.

Status: **Pass**, **Fail**, **Not run** (with reason).

ID	Feature	Scenario	Steps	Expected	Observed	Status
TC-01	Home	Visitor opens home	GET `/`	Objectives, roles, links render	Rendered (01-home.png)	Pass
TC-02	Navigation / responsive	Mobile width 390 px	Open `/`	Scrollable nav bar under header	Rendered (01-home-mobile.png)	Pass
TC-03	Registration form	Open sign-up	GET `/auth?mode=up`	Name, email, password, phone, age fields	Rendered (09-register.png)	Pass
TC-04	Login form	Open login	GET `/auth?mode=in`	Email, password	Rendered (08-login.png)	Pass
TC-05	Packages for visitors	Visitor opens `/plans`	GET page; REST `subscription_groups` with public key	Silver and Platinum cards	Empty list; API returns `permission denied for function has_role`	**Fail (defect D-01)**
TC-06	Payment methods for visitors	REST `payment_methods` with public key	—	Active methods listed	Same permission error	**Fail (D-01)**; checkout requires sign-in so impact limited to that page
TC-07	Posts visibility	REST `posts` with public key	—	Only approved posts, no error	`[]` (0 posts in DB), no error	Pass
TC-08	Guidance empty state	Visitor opens `/exercises`	—	"لا يوجد إرشاد متاح"	Rendered (02-exercises.png)	Pass
TC-09	Health calculator	Enter weight/height	Observe BMI	BMI and category	Page renders; calculator not exercised by script	Pass (render) / calculation reviewed in code
TC-10	Test quizzes	Open `/test`	—	Four quizzes	Rendered (04-test.png)	Pass
TC-11	Protected page gate	Visitor opens `/dashboard`, `/new`	—	"يرجى تسجيل الدخول"	Rendered (11, 13 screenshots)	Pass
TC-12	Admin gate	Visitor opens `/admin`	—	"هذه الصفحة للمشرف فقط"	Rendered (12-admin-signed-out.png)	Pass
TC-13	Capacity	Capacity 1; subscriber A subscribes	SQL in rolled-back transaction	Count = 1/1	Count 1/1	Pass (DB test)
TC-14	Full group	Subscriber B subscribes to full	same	`NO_SPACE_IN_GROUP` → "لا يوجد مكان في المجموعة"	Refused with NO_SPACE_IN_GROUP	Pass (DB test)

ID	Feature	Scenario	Steps	Expected	Observed	Status
		group				
TC-15	Duplicate registration	A subscribes again	same	`ALREADY_REGISTERED`	Refused	Pass (DB test)
TC-16	Rejection frees seat	Admin rejects A	same	Count 0	0	Pass (DB test)
TC-17	Deletion frees seat	Admin deletes subscription	same	Count 0	0	Pass (DB test)
TC-18	Capacity below count	Admin sets capacity < current	Admin page	Error message	Guard in `setCap` (code review)	Not run in UI (no accounts)
TC-19	Attendance without active subscription	Insert attendance	SQL test	`NOT_ACTIVE_IN_GROUP`	Refused	Pass (DB test)
TC-20	Attendance duplicate same day	Second insert	SQL test	`ATTENDANCE_ALREADY_RECORDED`	Refused	Pass (DB test)
TC-21	Attendance future date	Insert future date	SQL test	`OUTSIDE_GROUP_PERIOD`	Refused	Pass (DB test)
TC-22	Rejected content hidden	Rejected post, visitor read	SQL test	Not visible	Not visible	Pass (DB test)
TC-23	Content submission pending	Specialist inserts guidance	UI	Status pending	Trigger logic reviewed	Not run (no accounts)
TC-24	Resubmission	Author edits rejected item	UI	Pending, reason cleared	Trigger logic reviewed	Not run (no accounts)
TC-25	Admin approve/reject	Content review tab	UI	Status changes; reason required	Code reviewed	Not run (no accounts)
TC-26	Block member	Admin blocks, member posts	UI	Refused	Policies reviewed	Not run (no accounts)
TC-27	Questions & alerts	Ask, answer, alert, mark read	UI	Round-trip works	Code reviewed	Not run (no accounts)
TC-28	Email verification	Register then resend	UI	Email sent	Not run (would create a real account)	Not run

Defects found

- **D-01 — Packages and payment methods fail to load for visitors.** The read rules for `subscription_groups` and `payment_methods` call `has_role()`, but visitors are not allowed to run that function, so the whole request fails. Signed-in users are unaffected. The package page therefore appears empty until a visitor signs in. This documentation task did not change the application; fixing it is a small database change (see 16).

15 — Traceability Matrix

Requirement	Interface	Workflow (05)	Use case	Database entity	Source component	Test
FR-01 Register	`/auth?mode=up`	§1	UC01	auth user, `profiles`, `member_contacts`, `user_roles`	`auth.tsx` AuthPage; `handle_new_user`	TC-03
FR-02 Verify	`/verify`	§1	UC02	auth user	`verify.tsx`	TC-28
FR-03 Login	`/auth?mode=in`	§1	UC03	auth session, `user_roles`	`auth.tsx`, `lib/auth.tsx`	TC-04
FR-05 Default roles	—	§1	UC01	`user_roles`	`handle_new_user`	—
FR-06 Trainer role	`/admin` Users	§8	UC18	`user_roles`	`admin.tsx` `Users`	—
FR-07/08 Block	`/admin` Users	§8	UC18	`profiles.status`	`Users`; `protect_profile_status`; `is_active_member`	TC-26
FR-09 Profile	`/profile`	—	UC13	`profiles`, `member_contacts`	`profile.tsx`	—
FR-10 Packages	`/plans`	§2	UC07	`subscription_groups`	`plans.tsx`, `fetchGroups`	TC-05
FR-11 Capacity	`/plans`, `/checkout/$id`, `/dashboard`	§2	UC07	`subscription_groups`, `subscriptions`	`isFull`, `check_group_space`	TC-13, TC-14
FR-12 No duplicate	`/checkout/$id`	§2	UC07	`subscriptions`	`check_group_space`	TC-15
FR-13 Seat count	`/admin` Requests	§2	UC19	`subscription_groups.subscriber_count`	`sync_group_count`	TC-16, TC-17
FR-14 Payment request	`/checkout/$id`	§2	UC08	`payments`, `payment_methods`	`checkout.$id.tsx`	TC-06
FR-15 Accept/reject request	`/admin` Requests	§2	UC19	`subscriptions`, `payments`	`Requests.decide`	TC-16
FR-16 Groups & capacity	`/admin` Groups	§2	UC20	`subscription_groups`	`Groups`, `setCap`	TC-18
FR-17 Payment methods	`/admin` Methods	—	UC21	`payment_methods`	`Methods`	TC-06
FR-18 Record attendance	`/dashboard`	§3	UC09	`attendance`	`MyAttendance`, `validate_attendance`	TC-19–21
FR-20 Review attendance	`/dashboard` Trainer	§3	UC16	`attendance`	`ReviewAttendance`	—
FR-21 Manage attendance	`/admin` Attendance	§3	UC23	`attendance`	`AttendanceAdmin`	—
FR-22/23 Questions	`/dashboard`	§7	UC10, UC15	`queries`	`MyQueries`, `AnswerQueries`	TC-27
FR-24 Alerts	`/dashboard`	§7	UC12, UC17	`alerts`	`MyAlerts`, alert sender	TC-27
FR-25/26 Guidance	`/exercises`, `/dashboard`	§4	UC14	`devices`	`GuideForm`, `MyContent`	TC-08, TC-23

Requirement	Interface	Workflow (05)	Use case	Database entity	Source component	Test
FR-27 Submission pending	`/exercises`, `/new`	§4	UC14, UC11	`devices`, `posts`	`enforce_content_approval`	TC-23
FR-28–30 Review/approve/reject	`/admin` Content review	§5	UC22	`devices`, `posts`	`ContentReview.decide`	TC-22, TC-25
FR-31 Resubmission	`/exercises`, `/posts/$id`	§6	UC14	`devices`, `posts`	`enforce_content_approval`	TC-24
FR-32 Community	`/community`	§9	UC06	`posts`	`community.tsx`, `fetchPosts`	TC-07
FR-33 New post	`/new`	§9	UC11	`posts`	`new.tsx`	—
FR-34 Comments/likes	`/posts/$id`	§9	UC11	`comments`, `post_likes`	`posts.$id.tsx`	—
FR-35 Health	`/health`	—	UC04	—	`health.tsx` `Bmi`	TC-09
FR-36 Test	`/test`	—	UC04	—	`test.tsx` `QuizRunner`	TC-10
NFR-01 RTL	all	—	—	—	`__root.tsx`	TC-01
NFR-02 Responsive	all	—	—	—	`SiteHeader.tsx`	TC-02
NFR-03/04 DB enforcement	—	all	—	all tables	migrations	TC-13–22

16 — Limitations and Future Enhancements

Limitations (current state, 27 Sep 2026)

1. **Defect D-01:** visitors cannot load packages or payment methods (database refuses the role check for non-signed-in requests). The Packages page appears empty until signing in.
2. **No user accounts yet**, so signed-in flows have not been exercised end to end in the browser; they are verified by code review and database tests only.
3. **No real payments.** Checkout records a request and a pending payment; the admin confirms manually.
4. **No group capacity set** on the two seeded groups, so neither can become full until the admin sets one.
5. **No session timetable**; attendance is one record per group per day (the presentation does not describe a timetable).
6. **Subscribers cannot cancel** a subscription themselves; only admin rejection (UI) or deletion (database only) frees a seat.
7. **Rejected content** has no dedicated "resubmit" button; the author resubmits by editing.
8. **Rejection reason** is entered through a simple browser prompt.
9. **Author names are hidden** from other members ("عضو") because profiles are private to owner and admin.
10. **Health content is general education**, not medical advice; BMI/water results are rough estimates.
11. **No automated test suite** in the repository.
12. **Arabic only**; no language switch.
13. Two older platform security warnings about signed-in users executing helper functions remain.
14. Performance, load and accessibility conformance were not measured.

Future Enhancements — NOT CURRENTLY IMPLEMENTED

- Fix D-01 by making the visitor read rules for packages and payment methods not depend on the role check (or allow visitors to run it safely).
- Public author display names through a limited, privacy-safe view.
- Subscriber self-cancellation with seat release.
- Group session timetable linked to attendance.
- Integration with a real payment provider.
- Dedicated rejection dialog and resubmit action.
- Email notifications for approvals, answers and alerts.
- Automated unit and end-to-end tests.
- English interface option.
- Reporting dashboard for admin (attendance and subscription statistics).

17 — Academic Summary

Problem. Fitness information is scattered and unreviewed, and trainees lack one place to get correct guidance on exercises, equipment and sports nutrition, join a supervised group, and interact with a community under administrative oversight.

Analysis. Requirements were gathered and modelled for three actors — Subscriber, Specialist (trainer) and Admin — through use cases, activity diagrams (including the documented "check space in the group" decision and the admin content check), and eight key interfaces: Registration, Login, Verification, Home, Subscription, Payment, Health and Test.

Design. Object-oriented analysis produced the domain classes (profile, role, subscription group, subscription, payment, payment method, attendance, device guidance, query, alert, post, comment, like) and their associations. The database design normalised these into 14 related tables with primary/foreign keys, constraints and role-based access rules, keeping roles in a separate table and enforcing business rules (capacity, one registration per group, attendance validity, content approval) inside the database.

Implementation. An Arabic right-to-left web application built with React, TanStack Start, Tailwind CSS and Lovable Cloud (Postgres + Auth). It provides all eight documented interfaces plus role dashboards, an admin panel, trainer guidance with admin approval, and a community area.

Results. Every workflow from the analysis is implemented and protected at the database level; public pages were verified in the browser, and core rules were verified with rolled-back database tests. One visitor-facing defect (packages not loading when signed out) and several scope limitations are documented honestly in documents 14 and 16.

My role in the project.

- Requirements Analysis
- Object-Oriented Analysis & Design
- Database Design

The application code was produced with an AI-assisted development platform based on this analysis and design; this summary does not claim that I personally programmed the entire platform.

Documentation Audit

Date: 27 September 2026. No application code, design, content, database or favicon was changed by this documentation task; only files under `docs/` were added.

What was documented

Overview, requirements (with implementation status), roles, 16 interfaces, 9 workflows, 23 use cases (13 detailed), OOAD class model, database design with generated schema appendix, UML set, architecture, file structure and source code (with full-source appendix), UI/UX, security, 28 test cases, traceability matrix, limitations, future enhancements, academic summary.

Diagrams created (`diagrams/`, each as .mmd + .svg + .png)

use-case, class-diagram, erd, activity-authentication, activity-content-approval, activity-group-registration, activity-attendance, sequence-subscribe, sequence-content-review, sequence-attendance, state-content, state-subscription, state-attendance, architecture, component, deployment (16 diagrams).

Screenshots (`screenshots/`)

Home (desktop + mobile), Sport and food, Health, Test, Packages (desktop + mobile), Community, Documentation, Login, Registration, Verify, Dashboard/Admin/New post (signed-out state).

Source files analysed

`src/routes/*` (16 route files), `src/lib/auth.tsx`, `community.ts`, `fitness.ts`, `utils.ts`, `src/components/SiteHeader.tsx`, `PostCard.tsx`, `src/router.tsx`, `src/start.ts`, `src/styles.css`, `src/integrations/supabase/types.ts`, `package.json`, `AGENTS.md`, all 10 files in `supabase/migrations/`.

Database entities documented

profiles, member_contacts, user_roles, subscription_groups, subscriptions, payments, payment_methods, attendance, devices, queries, alerts, posts, comments, post_likes; enums app_role, post_type, post_category; functions has_role, is_active_member, handle_new_user, protect_profile_status, check_group_space, after_subscription_change, sync_group_count, validate_attendance, enforce_content_approval, update_updated_at_column; 54 access policies.

Workflows documented

Registration/login, group registration with capacity, attendance, specialist content submission, admin review/approval/rejection, resubmission, questions and alerts, user management, community participation.

Could not be verified

- **Signed-in screens and flows in the browser:** the project has no user accounts, so no session could be created; these are documented from code and database tests.
- **Email delivery** of confirmation links (not triggered to avoid creating real accounts).
- **Internal password hashing and token handling** of the managed auth service (outside the project code).
- **Presentation file contents** were not re-opened in this task; alignment relies on the project history and the in-app documentation page, which was built from it.
- **Performance and accessibility** levels (not measured).

Issue discovered during the audit

- **D-01:** visitors cannot load packages or payment methods (`permission denied for function has_role`). Documented in 04, 14 and 16; not fixed because this task was documentation-only.

Labels used

Inferred items, not-implemented items and future work are labelled explicitly ("Not implemented", "Not run", "Future Enhancements — NOT CURRENTLY IMPLEMENTED").