Humkaam — Project Documentation

An On-Demand Marketplace for Verified Professionals

Humkaam Engineering

2026-04-27

1. Executive Summary

Humkaam is a full-stack on-demand marketplace that connects clients with verified professionals across multiple service categories — electricians, plumbers, home tutors, home nurses, and more. The platform is composed of three independently deployable applications backed by a single Node.js/NestJS API and a MySQL database, with real-time chat over WebSockets and S3-compatible file storage (with a local-disk fallback for development).

The system supports the full lifecycle of an engagement: **discovery → contact → booking → service delivery → review**. Every professional must pass an admin verification step (CNIC + supporting documents) before becoming visible to clients, and every review can only be submitted after a booking has been explicitly marked complete in the in-app chat.

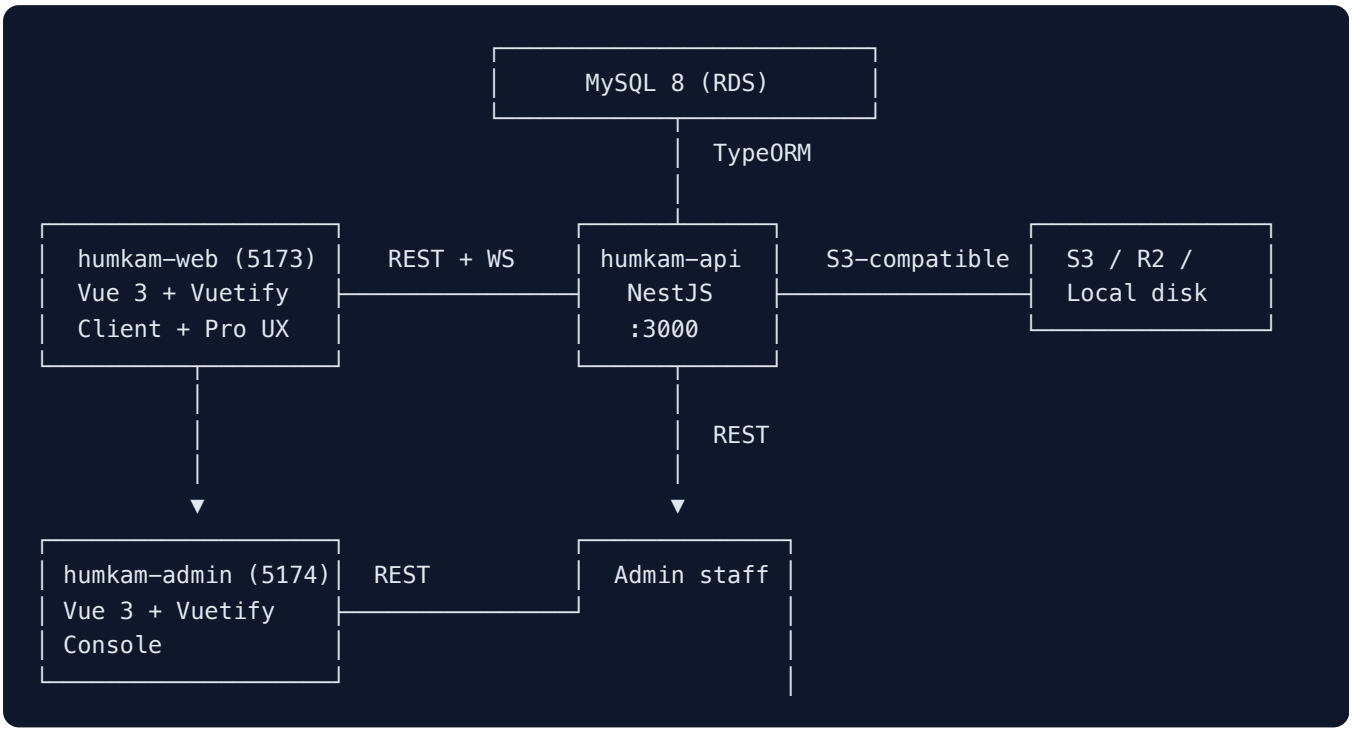
2. Project Goals

The primary objectives that shaped the design of Humkaam:

1. **Trust-by-default.** No professional appears in public search until a human admin has reviewed their CNIC and supporting documents and clicked “Approve & verify.”
2. **Friction-free discovery.** Clients must be able to go from the home page to a booking inside three clicks (category → professional → contact).
3. **Authentic reviews.** Reviews are gated server-side: the API rejects any `POST /reviews` unless the calling client has at least one booking with the target professional whose status is `completed`.
4. **Real-time chat.** Clients and professionals must be able to chat with message delivery, typing-state, presence indicators, and unread badges without polling.
5. **Single source of truth for staff.** A dedicated admin console gives staff a unified view of professionals, clients, bookings, conversations, reviews, categories, and a live KPI dashboard.
6. **Mobile-first responsive UI.** The client-facing site adapts cleanly from a wide desktop down to a 360 px phone, with a dark/light mode toggle.

3. System Architecture

The platform is composed of **three independent applications** plus a relational database and a file store:



FOLDER	PURPOSE	TECH
`humkam-api`	REST API + WebSocket gateway	NestJS 10, TypeORM, MySQL 8, Socket.io, Passport-JWT, bcrypt, Multer, AWS SDK v3
`humkam-web`	Public client / professional web app	Vue 3 (Composition API), Vite, Pinia, Vuetify 3, vue-router, axios, socket.io-client, vee-validate, CASL
`humkam-admin`	Internal admin console	Vue 3 + Vite + Pinia + Vuetify (same toolchain)

All three apps live as siblings under `ShiftSmart/`. They communicate **only** via the REST API and WebSocket gateway exposed by `humkam-api`. There is no direct DB access from either frontend.

4. Technology Stack

Backend (`humkam-api`)

- **NestJS 10** — modular HTTP/WS framework with dependency injection.
- **TypeORM 0.3** — ORM over MySQL 8 with `synchronize: true` in development; migrations expected before production.
- **Passport-JWT + @nestjs/jwt** — JWTs delivered as both an HttpOnly cookie and in the response body for cross-platform clients.
- **bcrypt** — password hashing (cost factor 12).
- **socket.io** + `@nestjs/websockets` — real-time chat, presence, booking updates.
- **Multer + @aws-sdk/client-s3** — multipart uploads with an S3 path and a local-disk fallback when AWS credentials are not present.
- **class-validator / class-transformer** — DTO-level validation and serialization.

Frontend (`humkam-web` , `humkam-admin`)

- **Vue 3 +** `<script setup>` **+ TypeScript**.
- **Vite** — dev server + build.
- **Pinia** — application state (auth, conversation, booking, presence, theme, category, professional, admin lists, stats, sidebar).
- **Vuetify 3** — component primitives (used sparingly; most UI is custom scoped CSS).
- **socket.io-client** — singleton with JWT auto-rebind on login/logout.
- **vue-router** with auth guards and per-route actor type enforcement.
- **CASL** — declarative ability checks per actor type (client, professional, admin, guest).

5. Data Model

The relational schema is owned by TypeORM entities under `humkam-api/src/*/models/`.

Tables

TABLE	PURPOSE
`users`	Professionals (the term “User” is reserved for the professional role)
`clients`	Clients who hire professionals
`admins`	Admin staff with console access
`conversations`	Chat thread between one client and one professional
`messages`	Individual chat messages (text / file / location)
`bookings`	Booking lifecycle row attached to a conversation
`reviews`	Client-authored reviews of professionals
`categories`	Service categories (Electrician, Plumber, etc.)

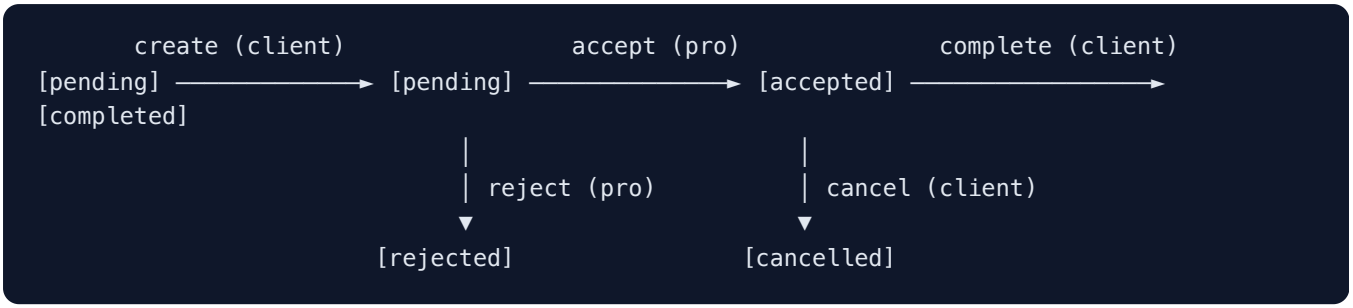
Key columns on `users` (the professional)

```
id, name, email (unique), phone, profile_picture, background_image,
password (excluded from serialization), profession, sub_profession, location,
skills (simple-array), about (text), rating (decimal 3,2),
verified (bool), featured (bool), featured_at (datetime),
cnic, documents (simple-array of URLs), verified_at,
availability (json — see below),
created_at, updated_at, deleted_at
```

`availability` is stored as a JSON object keyed by week-day:

```json
{
  "monday":    { "enabled": true,  "from": "09:00", "to": "17:00" },
  "tuesday":   { "enabled": true,  "from": "09:00", "to": "17:00" },
  "wednesday": { "enabled": true,  "from": "09:00", "to": "17:00" },
  "thursday":  { "enabled": true,  "from": "09:00", "to": "17:00" },
  "friday":    { "enabled": true,  "from": "09:00", "to": "17:00" },
  "saturday":  { "enabled": false, "from": "09:00", "to": "17:00" },
  "sunday":    { "enabled": false, "from": "09:00", "to": "17:00" }
}
```

Booking state machine



Reviews are only allowed when the corresponding booking has reached `completed`.

6. Authentication & Authorization

Three actor types

- **Professional** (`type: 'professional'`) — owns a `users` row.
- **Client** (`type: 'client'`) — owns a `clients` row.
- **Admin** (`type: 'admin'`) — owns an `admins` row, scoped to the admin app.

JWT payload

```ts
interface JwtPayload {
  id: number;
  type: 'professional' | 'client' | 'admin';
  email: string;
}
```

Tokens are signed with `JWT_SECRET`, expire in 7 days, and are returned in two ways:

1. As an HttpOnly cookie (used automatically by the web frontend).
2. Inline in the JSON body (used by the mobile app developer's client).

Guards

`humkam-api/src/auth/` ships four guards:

- `AuthGuard` — accepts any logged-in actor.
- `ProfessionalGuard` — only `type === 'professional'`.
- `ClientGuard` — only `type === 'client'`.
- `AdminGuard` — only `type === 'admin'`.

Routes that need finer-grained checks (e.g. "only the client who owns this conversation can post a message") delegate to service-layer ownership assertions.

Default seeded admin

An `OnApplicationBootstrap` hook in `AdminService` seeds a default admin on first launch using bcrypt-hashed credentials. The seeded values can be overridden via environment variables before first deployment to production.

7. Public Web App (`humkam-web`)

7.1 Home page

Composed of seven section components in `src/views/landing/sections/`:

1. **HeroSection** — animated gradient background, mouse parallax floating cards, glass-morphism search bar (single keyword input + CTA), trust-signal row.
2. **Categories grid** (in `Home.vue`) — driven by the `/api/categories` endpoint with the local hard-coded list as a fallback. Two-column grid on mobile, multi-column on desktop, hover lifts.
3. **HowItWorks** — three-step explainer (Search → Chat → Done) with a numbered badge and connector arrows.
4. **TopProfessionals** — fetched from `/api/professionals?featured=true` so admin-curated pros populate the carousel.
5. **PromoBanner** — gradient card with concentric spinning rings.
6. **Testimonials** — auto-rotating carousel with prev/next + dot navigation.
7. **JoinCta** — split CTA cards: “I need help” / “I’m a professional”.

All sections fade in on scroll via `IntersectionObserver` and respect `prefers-reduced-motion`.

7.2 Auth pages

Four pages share a single `AuthSplitLayout.vue` with four distinct hero artworks:

PAGE	HERO THEME
Client login	Recent-chats feed mock
Client signup	Discovery orbit with category chips
Professional login	Live dashboard with animated bar chart
Professional signup	Profile-builder with typewriter name + stamp

Each form lives on the right; the artwork sits on the left and respects mouse-parallax depth.

7.3 Professional signup (4 steps)

1. **Account** — name, email, password.
2. **Profession + location** + optional CNIC text.
3. **Profile picture** (optional, can skip).
4. **Verification documents** (optional, multi-file: CNIC scan, certifications).

The pro is created on step 2 with `verified = false`; steps 3 and 4 attach artifacts and do not block account creation. After step 4 the user lands on their portfolio.

7.4 Search & discovery

`/search` filters verified professionals by keyword, profession, and location. Each card shows avatar, name, profession/sub-profession, location, and a verified pill. Clicking opens the public profile.

7.5 Public professional profile

Layout: hero card with cover + avatar + name + verified badge + rating chip; quick-stats strip (rating, reviews, working days/week); two-column body with **About**, **Skills**, **Reviews**, **Weekly availability** (working/off breakdown), and **Contact**. Includes a “Start a conversation” dialog.

7.6 Pro portfolio (logged-in pro)

Same hero/stats/grid as the public view, **plus pencil icons on every section** (Identity, About, Skills, Contact, Availability) and camera buttons on the avatar and cover. Each pencil opens a focused single-purpose modal.

8. Real-Time Chat

8.1 Server (`chat.gateway.ts`)

- JWT-authenticated handshake (token from `auth.token` or `Authorization` header).
- On connect, every authenticated socket auto-joins a personal room `user_<type>_<id>` and registers in an in-memory presence map (per-user open socket count → supports multiple tabs).
- Conversation rooms are `conversation_<id>`.
- Events emitted:
  - `new_message` (to both `conversation_<id>` and the two personal rooms, so recipients receive even when not in chat page).
  - `message_deleted`.
  - `booking_update`.
  - `presence_changed { type, id, online }` — broadcast on first connect / last disconnect.
  - `presence_snapshot` — full snapshot, sent on connect and on `request_presence` to defeat the listener-attach race.

8.2 Client (`humkam-web/src/stores/conversation.ts` + `presence.ts`)

- Singleton socket with JWT-aware reconnect: the socket is torn down and rebuilt whenever `auth.jwt` changes (login / logout / re-login).
- `conversation` store keeps `unread_count` per conversation and exposes `totalUnread` for the header badge. Self-echoes are filtered.
- `presence` store maintains a `Set` of `<type>_<id>` keys; the chat header derives `isPartnerOnline` from it (green pulsing dot when online, red when offline).
- Header **Messages** link shows a red gradient pill with the total unread count; it disappears the moment all conversations are read.

8.3 Read receipts

- `GET /api/conversations/:id/messages` automatically marks the requested conversation as read for the caller.
- `POST /api/conversations/:id/read` is also available (used when a new message arrives while the user is already viewing the conversation, so the server's count stays at 0 and the badge does not re-appear after a refresh).

9. Booking Lifecycle

`humkam-api/src/booking/`

- `POST /api/bookings` — client only. Creates a booking attached to an existing conversation in `pending` state.
- `POST /api/bookings/:id/accept` — pro only. `pending → accepted`, sets `accepted_at`.
- `POST /api/bookings/:id/reject` — pro only. `pending → rejected`.
- `POST /api/bookings/:id/complete` — client only. `accepted → completed`, sets `completed_at`. After this, the client becomes review-eligible and the in-chat **review dialog** auto-opens.
- `POST /api/bookings/:id/cancel` — client only.
- `GET /api/bookings?conversation_id=...` — list bookings on a conversation.

Each transition emits a `booking_update` socket event so both parties' UIs update in real time without a round-trip.

10. Review System

- `ReviewService.create()` — throws `ForbiddenException` unless `BookingService.hasCompletedBetween(clientId, professionalId)` returns `true`. Server-enforced, never relaxed by the frontend.
- After a successful insert, the professional's average rating is recomputed via `AVG(rating) WHERE professional_id = ?` and stored on the user row.
- `GET /api/reviews/can-review/:professionalId` — returns `{ allowed: boolean }`; the web UI uses it to gate the in-chat review dialog.
- The standalone "Write a review" form on the public profile has been **removed**; the only entry point is the chat dialog that auto-opens on "Mark Complete." This prevents drive-by reviews.

11. Verification & Top-Rated

11.1 Verification

- New pros default to `verified = false`. Public search and `findOne` filter to `verified = true` so unverified pros are not discoverable.
- Admin sees a **Pending verification** tab on the professionals list.
- The admin detail page exposes:
  - **CNIC** input + save (`PUT /api/admin/professionals/:id/cnic`).
  - **Documents** uploader / per-file remove (`POST/DELETE /api/admin/professionals/:id/documents`).
  - **Approve & verify** / **Revoke verification** confirm-dialog pair (`PUT /api/admin/professionals/:id/verify`).
- A pro can also self-upload during signup step 4 and from their portfolio via `POST /api/users/me/documents`.

11.2 Top-Rated / Featured

- A separate `featured: boolean` flag on the user row, controlled only by the admin (`PUT /api/admin/professionals/:id/featured`).
- The feature endpoint refuses to feature an unverified professional (returns 400). This guarantees the home-page carousel always shows verified people.
- Public `/api/professionals?featured=true` returns featured pros ordered by `featured_at DESC, rating DESC`. The home page's TopProfessionals component hits this endpoint.

12. Admin Console (`humkam-admin`)

Sky-blue sidebar with collapsible “minimize” toggle (hamburger icon), gold accent for active states. Top-right header has a sun/moon dark-mode toggle and a logout button (with confirm dialog).

12.1 Dashboard

- KPI cards: Total Professionals, Total Clients, Total Bookings (with completed / pending pills), Reviews + average rating, Conversations, Booking Breakdown (chip strip of pending / accepted / completed / rejected / cancelled).
- Two-column row: **Top Professionals** card (clickable → ranked list) and **Recent Signups** (latest pros + clients side-by-side).
- A refresh button forces a re-fetch.

12.2 Professionals module

- List with three tabs: **All**, **Pending verification**, **Top rated**.
- Search, paginate, soft-delete from the row.
- “Add professional” button → `CreateProfessional.vue` form.
- Click a row → `ProfessionalDetail.vue` (header card with verified + featured pills, CNIC panel, documents panel, About + Skills, Approve/Verify, Make Top-rated, Edit, Delete).
- Edit goes to `EditProfessional.vue` (same form, pre-filled).

12.3 Categories module

Card-grid list with cover/icon/sort_order pill. Create + Edit reuse `CategoryForm.vue` (auto-slug from label, live image preview, sort order, Font Awesome icon class). Categories created here flow immediately to the public site via `useCategoryStore.load()`.

12.4 Other modules

- **Clients** — searchable table; click row → ClientDetail; soft-delete.
- **Bookings** — status-filter tabs (All/Pending/Active/Completed/Rejected/Cancelled), shows client + professional + timestamps.
- **Conversations** — read-only audit table.
- **Reviews** — card grid with star ratings; **Edit review** view (separate file) lets admin adjust rating/comment, server recomputes pro’s average; **Delete review** also recomputes.

13. File Uploads

`humkam-api/src/s3/s3.service.ts` provides a single `uploadFile(file, folder)` method that:

1. Uses S3 if `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, `AWS_REGION`, and `S3_BUCKET_NAME` env vars are set. Returns the public URL.
2. Otherwise falls back to writing the file under `./uploads/<folder>/` and serves it via `app.use('/files/uploads', express.static(...))` in `main.ts`. Returns `APP_API_SERVER + '/files/uploads/<filename>'`.

Endpoints that accept files (multipart/form-data):

- `POST /api/users/me/profile-image`
- `POST /api/users/me/background-image`
- `POST /api/users/me/documents` (max 10 files)
- `POST /api/clients/me/profile-image`
- `POST /api/conversations/:id/messages/document`
- `POST /api/admin/professionals/:id/documents`
- `POST /api/files/generic`

14. Theming & Dark Mode

- Both web and admin ship a Pinia `theme` store (`light` / `dark`) that persists to `localStorage` (`theme` for web, `admin_theme` for admin) and initially honours `prefers-color-scheme: dark` if no preference is saved.
- The store toggles a `dark` class on `<html>`. All dark styles are written as `html.dark .selector { ... }` overrides.
- Coverage:
  - **Web**: body, header, hero, categories, search bar, top-pros, promo, testimonials, profile pages, chat (sidebar / messages / input / booking banner / review modal), confirm dialogs, auth pages.
  - **Admin**: layout shells, topbar, KPI cards, panels, tables, status pills, tabs, search boxes, ConfirmDialog, paginator, review cards, the admin login screen.
- The toggle button in both apps is a sun/moon icon that rotates 20° on hover and uses gold tints in dark mode.

15. API Reference (Public + Authed)

A complete, importable Postman collection ships with the repo at `Humkaam-API.postman_collection.json`. Highlights:

Auth

METHOD	PATH	NOTES
POST	`/api/auth/professional/register`	Atomic 3-step signup; sets cookie
POST	`/api/auth/professional/login`	
DELETE	`/api/auth/professional/logout`	
POST	`/api/auth/client/register`	
POST	`/api/auth/client/login`	
DELETE	`/api/auth/client/logout`	
POST	`/api/auth/admin/login`	
DELETE	`/api/auth/admin/logout`	
GET	`/api/auth/check`	`{ authenticated, id, type, email }`

Public

METHOD	PATH
GET	`/api/categories`
GET	`/api/professionals` (search, filters)
GET	`/api/professionals/:id`
GET	`/api/professionals/:id/reviews`
GET	`/api/health`

Authenticated user-side

METHOD	PATH
GET	`/api/users/me`
PUT	`/api/users/me`
POST	`/api/users/me/profile-image`
POST	`/api/users/me/background-image`
POST	`/api/users/me/documents`
DELETE	`/api/users/me/documents`
GET	`/api/clients/me`
PUT	`/api/clients/me`
POST	`/api/clients/me/profile-image`

Conversations & messages

METHOD	PATH
POST	`/api/conversations`
GET	`/api/conversations`
GET	`/api/conversations/:id/messages`
POST	`/api/conversations/:id/messages`
POST	`/api/conversations/:id/messages/document`
DELETE	`/api/conversations/:id/messages/:messageId`
DELETE	`/api/conversations/:id`
POST	`/api/conversations/:id/read`

Bookings

METHOD	PATH
POST	`/api/bookings`
POST	`/api/bookings/:id/accept`
POST	`/api/bookings/:id/reject`
POST	`/api/bookings/:id/complete`
POST	`/api/bookings/:id/cancel`
GET	`/api/bookings?conversation_id=`

Reviews

METHOD	PATH
POST	`/api/reviews`
GET	`/api/reviews/can-review/:professionalId`

Admin

METHOD	PATH
GET	`/api/admin/me`
PUT	`/api/admin/me`
GET	`/api/admin/stats`
GET	`/api/admin/stats/top-professionals`
GET	`/api/admin/stats/recent-signups`
GET	`/api/admin/professionals` (search, verified, page)
GET	`/api/admin/professionals/:id`
POST	`/api/admin/professionals`
PUT	`/api/admin/professionals/:id`
PUT	`/api/admin/professionals/:id/verify`
PUT	`/api/admin/professionals/:id/featured`
PUT	`/api/admin/professionals/:id/cnic`
POST	`/api/admin/professionals/:id/documents`
DELETE	`/api/admin/professionals/:id/documents`
DELETE	`/api/admin/professionals/:id`
GET	`/api/admin/clients` (search, page)
GET	`/api/admin/clients/:id`
DELETE	`/api/admin/clients/:id`
GET	`/api/admin/bookings` (status, page)
GET	`/api/admin/conversations` (page)
GET	`/api/admin/reviews` (page)
GET	`/api/admin/reviews/:id`
PUT	`/api/admin/reviews/:id`
DELETE	`/api/admin/reviews/:id`
GET	`/api/admin/categories`
GET	`/api/admin/categories/:id`

METHOD	PATH
POST	`/api/admin/categories`
PUT	`/api/admin/categories/:id`
DELETE	`/api/admin/categories/:id`

16. Security Posture

- All passwords are bcrypt-hashed with cost factor 12.
- JWT delivered as `HttpOnly` cookie (`SameSite=Lax` , `Secure` in production).
- All write endpoints require an authenticated guard; many require a specific actor type.
- DTOs validated with `class-validator` ; unknown fields rejected only where appropriate (`forbidUnknownValues: false` is set globally to avoid breaking clients that include extras).
- CORS whitelist is explicit and centred on the dev URLs (`5173` , `5174` , `3000` , `8080`); it must be updated to the production origins before deploy.
- Soft-deletes on professionals, clients, and messages preserve audit trails.
- Admins are intentionally **forbidden** from sending messages on behalf of users (`ConversationService.asSender` throws if `payload.type === 'admin'`).
- Reviews can only be submitted post-completion (server-enforced, never client-enforced).

Pre-production checklist

1. Disable TypeORM `synchronize: true` and run real migrations.
2. Rotate the seeded admin credentials via env vars or first-login change.
3. Set a strong `JWT_SECRET` .
4. Configure S3 (`AWS_*` , `S3_BUCKET_NAME`) so files survive container restarts.
5. Update the CORS whitelist in `main.ts` with production domains.
6. Set `NODE_ENV=production` so cookies are issued as `Secure` .
7. Set `VITE_API_URL` and `VITE_SOCKET_URL` at frontend build time.

17. Deployment

The recommended starter stack (low cost, fast to ship):

PIECE	SERVICE
API + WebSockets	**Railway** or **Render** ($5/mo)
MySQL	Railway MySQL or PlanetScale free tier
File store	Cloudflare R2 (S3-compatible) or AWS S3
Web frontend	Vercel
Admin frontend	Vercel (separate project)
Domain + HTTPS	Cloudflare or Namecheap

The full-AWS path (more setup, more cost):

PIECE	SERVICE
API	Elastic Beanstalk or ECS Fargate
MySQL	RDS for MySQL (db.t3.micro)
File store	S3
Frontends	S3 + CloudFront (or Amplify)
Domain + HTTPS	Route 53 + ACM

18. Future Work

- **Push notifications** for unread messages when the web app is closed.
- **Server-sent typing indicator** (“Ali is typing…”) using a debounced socket event.
- **Rate-limited “Report” action** on reviews and conversations to feed an admin moderation queue.
- **Calendar integration** so a booking’s accepted slot appears on the professional’s external calendar.
- **Stripe / payment gateway integration** to handle payment-on-completion.
- **Localisation (Urdu)** — string extraction is already trivial since most copy is inline; only the routes need translation tables.
- **Mobile applications** consuming the same REST + WebSocket layer (the Postman collection ships with this in mind).

19. Conclusion

Humkaam demonstrates a complete vertical slice of a verified-professional marketplace: trust on the supply side via admin-gated verification, friction- free discovery on the demand side, real-time chat with presence and unread indicators, a booking lifecycle that gates reviews to completed work, and a fully functional admin console for staff oversight. The codebase is split along clean module boundaries (NestJS modules on the backend, Pinia stores + view modules on the frontends), and dark mode, mobile responsiveness, and internationalisation hooks are all in place for the next iteration.

Appendix A: Repository Layout

```
ShiftSmart/
├── humkam-api/                 NestJS backend
│   └── src/
│       ├── admin/              admin module + stats + list services
│       ├── auth/               JWT + guards
│       ├── booking/            booking lifecycle
│       ├── category/           categories CRUD
│       ├── chat/               Socket.io gateway
│       ├── client/             client profile
│       ├── conversation/       conversations + messages
│       ├── permission/         CASL abilities
│       ├── professional/       public discovery
│       ├── review/             review CRUD with completion gate
│       ├── s3/                 S3 / local-disk file service
│       ├── upload/             generic uploaders
│       └── user/               professional profile
│
├── humkam-web/                 client / professional frontend
│   └── src/
│       ├── components/         shared UI primitives + auth + profile
│       ├── composables/        useSocket, useFileUpload
│       ├── config/             categories fallback
│       ├── layouts/            blank + full
│       ├── models/             TS interfaces
│       ├── plugins/            vuetify, casl
│       ├── router/             MainRoutes + AuthRoutes
│       ├── stores/             auth, conversation, presence, theme,
│       │                       booking, professional, category
│       └── views/
│           ├── authentication/
│           ├── landing/        home + section components
│           ├── modules/        category, professional, chat
│           └── profile/        MyPortfolio, ProfessionalProfile, ClientProfile
│
└── humkam-admin/               admin console
    └── src/
        ├── components/
        ├── layouts/
        ├── router/
        ├── stores/             admin auth, adminLists, stats, theme,
        │                       categories, sidebar
        └── views/
            ├── authentication/  AdminLogin
            ├── dashboard/
            └── modules/
                ├── bookings/
                ├── categories/
                ├── clients/
                ├── conversations/
```

```
├── professionals/   list + detail + create + edit
└── reviews/         list + edit
```

Appendix B: Default Credentials (development only)

ROLE	EMAIL	PASSWORD
Admin	`humkam@gmail.com`	`thisthat`

The admin row is auto-seeded by `AdminService.onApplicationBootstrap` if `humkam@gmail.com` does not exist in the `admins` table. **Change immediately in any non-development environment.**