Freela — პროექტის სრული აღწერა (Architecture + Structure)

დოკუმენტი აღწერს Freela-ს მიმდინარე მდგომარეობას: არქიტექტურა, მონაცემთა მოდელი, ძირითადი ფლოუები, API-ები, UI გვერდები და საქაღალდე/ფაილების სტრუქტურა.

> *შენიშვნა: უსაფრთხოების მიზნით **არ** არის ჩასმული `.env.local` /სეკრეტების შიგთავსი. ასევე, დოკუმენტი არ ბეჭდავს თითოეული ფაილის სრულ კოდს (ეს არის რეპოზიტორიაში).*

1) მაღალი დონის არქიტექტურა

Framework: Next.js 16 (App Router)
Auth: NextAuth (Credentials) + Prisma users + bcrypt
DB: Postgres (Docker Compose Dev)
ORM: Prisma v7 (migrations: SQL files + Prisma CLI)
UI: React + Tailwind + მცირე UI კომპონენტები (`Card` , `Button` , `Badge` , `Container`)
Realtime: SSE (`/api/realtime`) + in-memory connection map (dev/proc single instance)

Request flow (App Router)

- UI pages (`src/app/**/page.tsx`) = Server Components (default) → server-side data fetch via Prisma + server-side guards (NextAuth session).
- Client components (formები, SSE subscribe) = `"use client"` → fetch API routes, `router.push/refresh` .

Auth & Access Control

- `src/lib/auth.ts` defines NextAuth options.
- Session contains `session.user.id` and `session.user.role` .
- Server-side guards: `getServerSession(authOptions)` + role checks + redirects/404.
- API routes also enforce auth/role and ownership constraints.

2) მონაცემთა მოდელი (Prisma)

Prisma schema: `prisma/schema.prisma`

ძირითადი ცხრილები

- `User` — (id, name, email, role, passwordHash, createdAt/updatedAt)
- `Profile` — freelancer პროფილი (title, bio, skills, hourlyGEL, ...)
- `Project` — employer-ის პროექტი (title, description, city, budgetGEL, isOpen, ...)
- `Proposal` — freelancer განაცხადი პროექტზე (unique: projectId+freelancerId) + `status` (PENDING/ACCEPTED/REJECTED)
- `Thread` — პროექტზე საუბრის "თრედი" (unique: projectId+freelancerId)
- `Message` — შეტყობინება თრედში (senderId, body, createdAt)
- `Notification` — ინ-აპ ნოტიფიკაციები (type, title, body, href, readAt, createdAt)

Enums

- `Role` : `EMPLOYER | FREELANCER | ADMIN`
- `ProposalStatus` : `PENDING | ACCEPTED | REJECTED` (immutable გადაწყვეტილება)
- `NotificationType` : `MESSAGE | PROPOSAL_STATUS | NEW_PROPOSAL`

3) ფუნქციონალი (მოკლე ჩამონათვალი)

3.1 რეგისტრაცია/ლოგინი

- `/api/register` — ქმნის user-ს bcrypt hash-ით, role mapping, ქმნის Profile-ს (თუ საჭიროა).
- NextAuth Credentials login — Prisma user lookup + bcrypt compare.

3.2 პროექტები (EMPLOYER)

- `/projects/new` — ახალი პროექტის შექმნა (EMPLOYER only)
- `/dashboard/projects` — საკუთარი პროექტების სია + შეთავაზებების count
- `/dashboard/projects/[id]` — პროექტის დეტალები + proposals list + Accept/Reject + "ფრილანსერს მიწერა"

3.3 პროექტები (FREELANCER)

- `/projects` — პროექტების directory + filter/sort/pagination URL sync
- `/projects/[id]` — პროექტის დეტალი + Apply (თუ proposal არ არსებობს) ან სტატუსი + "დამსაქმებელს მიწერა"

3.4 პროფილი (FREELANCER)

- `/dashboard/profile` — პროფილის ნახვა/რედაქტირება (PATCH `/api/profile`)

3.5 შეტყობინებები (Messaging)

- `/dashboard/messages` — threads list
- `/dashboard/messages/[id]` — thread დეტალი + message list + composer (მხოლოდ აქ არის input)
- Thread create rule: **Thread იქმნება მხოლოდ თუ არსებობს Proposal (projectId, freelancerId)** და creator არის მონაწილე.

3.6 Notifications

- `/dashboard/notifications` — ნოტიფიკაციების სია + "ყველას მონიშვნა წაკითხულად"
- Header badge (SSE რეალთაიმი)

3.7 Realtime (SSE)

- `/api/realtime` — EventSource stream:
  - `event: message` → `{ threadId, message }`
  - `event: notification` → payload (badge increment)

4) API Routes (სია)

ქვემოთ ჩამონათვალი არის რეალური ფაილებით:

- Auth:
  - `src/app/api/auth/[...nextauth]/route.ts`
- Register / Profile:
  - `src/app/api/register/route.ts`
  - `src/app/api/profile/route.ts`
- Projects / Proposals:
  - `src/app/api/projects/route.ts` (GET filters + POST create)
  - `src/app/api/projects/[id]/apply/route.ts`
  - `src/app/api/proposals/[id]/route.ts` (PATCH accept/reject + notifications)

- Threads / Messages:
  - `src/app/api/threads/route.ts` (GET list)
  - `src/app/api/threads/create/route.ts` (POST create/find thread; proposal required; participant required)
  - `src/app/api/threads/[id]/messages/route.ts` (GET/POST messages; auto-create guarded)
- Notifications:
  - `src/app/api/notifications/route.ts` (GET list + unreadCount)
  - `src/app/api/notifications/[id]/route.ts` (PATCH mark read)
  - `src/app/api/notifications/mark-all-read/route.ts` (POST mark all read)
- Freelancers directory:
  - `src/app/api/freelancers/route.ts` (GET filters/sort/pagination)
- Realtime (SSE):
  - `src/app/api/realtime/route.ts`

5) UI Pages (სია)

- Public:
  - `src/app/page.tsx`
  - `src/app/projects/page.tsx`, `src/app/projects/[id]/page.tsx`
  - `src/app/freelancers/page.tsx`, `src/app/freelancers/[id]/page.tsx`
  - `src/app/about/page.tsx`, `src/app/pricing/page.tsx`, `src/app/contact/page.tsx`
  - Legal: `src/app/legal/terms/page.tsx`, `src/app/legal/privacy/page.tsx`
- Auth:
  - `src/app/auth/login/page.tsx`, `src/app/auth/register/page.tsx`
- Dashboard:
  - `src/app/dashboard/page.tsx`
  - `src/app/dashboard/profile/page.tsx`
  - `src/app/dashboard/projects/page.tsx`, `src/app/dashboard/projects/[id]/page.tsx`
  - `src/app/dashboard/proposals/page.tsx`
  - `src/app/dashboard/messages/page.tsx`, `src/app/dashboard/messages/[id]/page.tsx`
  - `src/app/dashboard/notifications/page.tsx`
- Admin:
  - `src/app/admin/page.tsx`

6) Realtime/Events (სერვერული emit-ები)

- Message create (`POST /api/threads/[id]/messages`):
  - DB-ში ქმნის Message-ს
  - Update Thread.updatedAt
  - ქმნის Notification-ს recipient-სთვის (`type=MESSAGE`)
  - SSE emit: `message` ორივე მონაწილეზე + `notification` recipient-ზე
- Proposal status change (`PATCH /api/proposals/[id]`):
  - თუ status != PENDING → 409
  - Update status
  - Notification freelancer-ზე (`type=PROPOSAL_STATUS`)
  - SSE emit: `notification` freelancer-ზე

- Proposal submit (`POST /api/projects/[id]/apply`):
  - Notification employer-ზე (`type=NEW_PROPOSAL`)
  - SSE emit: `notification` employer-ზე

7) Environment / Config

Required env vars

Template: `.env.example`

- `NEXTAUTH_URL`
- `NEXTAUTH_SECRET`
- `DATABASE_URL` (Postgres)

Prisma config

- `prisma.config.ts` ლოდავს `.env` / `.env.local` და აწვდის `DATABASE_URL` -ს Prisma CLI-ს.

Dev DB

- `docker-compose.yml` — Postgres dev instance
- npm scripts: `db:up` , `db:down` , `db:reset`

8) Migrations

მიგრაციები ინახება `prisma/migrations/**/migration.sql` .

რეკომენდირებული გამოყენება (dev):

1. `npm run db:up`
2. `.env.local` -ში `DATABASE_URL` შეავსე Postgres-ზე
3. `npx prisma migrate dev --config=./prisma.config.ts`
4. `npm run prisma:generate`

Deploy notes: `docs/DEPLOY.md`

9) საქაღალდე/ფაილების სტრუქტურა (Index)

ქვემოთ არის repository-ს ძირითადი სტრუქტურა (გამოტოვებულია `.next/` , `node_modules/` , `dev.db` , სეკრეტები).

```
.
├─ docker-compose.yml
├─ docs/
│  ├─ DEPLOY.md
│  └─ PROJECT_OVERVIEW.md
├─ prisma/
│  ├─ schema.prisma
│  └─ migrations/
│     ├─ migration_lock.toml
│     ├─ 20260131213523_init/migration.sql
│     ├─ 20260131175452_role_employer/migration.sql
│     ├─ 20260131221721_role_client_to_employer/migration.sql
│     ├─ 20260131223056_project_fields_employer/migration.sql
```

```
│     ├─ 20260131224621_profile_skills_hourly/migration.sql
│     ├─ 20260131225543_proposal/migration.sql
│     ├─ 20260131231012_proposal_status/migration.sql
│     ├─ 20260131234510_threads_messages/migration.sql
│     └─ 20260131235910_notifications/migration.sql
├─ public/
│  ├─ logo.svg
│  └─ mark.svg
├─ scripts/
│  └─ check-env.mjs
├─ src/
│  ├─ app/
│  │  ├─ api/
│  │  │  ├─ auth/[...nextauth]/route.ts
│  │  │  ├─ freelancers/route.ts
│  │  │  ├─ notifications/route.ts
│  │  │  ├─ notifications/[id]/route.ts
│  │  │  ├─ notifications/mark-all-read/route.ts
│  │  │  ├─ profile/route.ts
│  │  │  ├─ projects/route.ts
│  │  │  ├─ projects/[id]/apply/route.ts
│  │  │  ├─ proposals/[id]/route.ts
│  │  │  ├─ realtime/route.ts
│  │  │  ├─ register/route.ts
│  │  │  ├─ threads/route.ts
│  │  │  ├─ threads/create/route.ts
│  │  │  └─ threads/[id]/messages/route.ts
│  │  ├─ auth/
│  │  │  ├─ login/page.tsx
│  │  │  ├─ login/login-form.tsx
│  │  │  ├─ register/page.tsx
│  │  │  └─ register/register-form.tsx
│  │  ├─ dashboard/
│  │  │  ├─ page.tsx
│  │  │  ├─ profile/page.tsx
│  │  │  ├─ profile/profile-form.tsx
│  │  │  ├─ projects/page.tsx
│  │  │  ├─ projects/[id]/page.tsx
│  │  │  ├─ projects/[id]/proposal-actions.tsx
│  │  │  ├─ projects/[id]/start-chat-button.tsx
│  │  │  ├─ proposals/page.tsx
│  │  │  ├─ messages/page.tsx
│  │  │  ├─ messages/message-composer.tsx
│  │  │  ├─ messages/[id]/page.tsx
│  │  │  └─ messages/[id]/message-thread.tsx
│  │  │  ├─ notifications/page.tsx
│  │  │  └─ notifications/notifications-list.tsx
│  │  ├─ projects/
│  │  │  ├─ page.tsx
│  │  │  ├─ projects-filters.tsx
│  │  │  ├─ new/page.tsx
│  │  │  ├─ new/new-project-form.tsx
```

```
│  │  │  └ [id]/page.tsx
│  │  │  └ [id]/apply-form.tsx
│  │  │  └ [id]/start-chat-button.tsx
│  │  ├ freelancers/
│  │  │  ├ page.tsx
│  │  │  ├ freelancers-filters.tsx
│  │  │  └ [id]/page.tsx
│  │  ├ admin/page.tsx
│  │  ├ layout.tsx
│  │  ├ globals.css
│  │  ├ not-found.tsx
│  │  ├ robots.ts
│  │  └ sitemap.ts
│  ├ components/
│  │  ├ site-header.tsx
│  │  ├ site-footer.tsx
│  │  ├ notifications/realtime-badge.tsx
│  │  ├ auth/auth-buttons.tsx
│  │  ├ auth/session-provider.tsx
│  │  ├ forms/lead-form.tsx
│  │  └ ui/{badge,button,card,container}.tsx
│  ├ lib/
│  │  ├ auth.ts
│  │  ├ prisma.ts
│  │  ├ realtime.ts
│  │  ├ role.ts
│  │  ├ proposal-status.ts
│  │  ├ search-params.ts
│  │  ├ site.ts
│  │  └ utils.ts
│  └ types/next-auth.d.ts
├ .env.example
├ eslint.config.mjs
├ next.config.mjs
├ package.json
├ package-lock.json
├ postcss.config.mjs
├ prisma.config.ts
├ tailwind.config.ts
└ tsconfig.json
```

10) სწრაფი “როგორ დავრბინოთ”

```
npm install
npm run db:up
# .env.local შევსება .env.example-ის მიხედვით
npx prisma migrate dev --config=./prisma.config.ts
npm run prisma:generate
npm run dev
```