Coony – Visão Geral do Projeto

Documentação técnica do aplicativo social-esportivo construído sobre Django. Este resumo cobre o stack, modelos de dados, rotas backend, superfícies de interface, assets estáticos e instruções de execução.

Stack e Dependências Principais

- **Backend:** Django 5.2.8, Python 3.x
- **Banco de dados:** SQLite por padrão (`dj-database-url` permite `DATABASE_URL` em produção)
- **Detecção de dispositivo:** `django-user-agents` decide entre views desktop/mobile
- **Frontend:** Google Material Symbols, Poppins/Montserrat/Roboto, CropperJS (perfil), emoji-picker web component, scripts próprios em `usuarios/static/script`
- **Outros utilitários:** Toast system custom (HTML/CSS/JS), `db_viewer.py` (Tkinter) para inspecionar `db.sqlite3`

Estrutura do Projeto

Caminho	Descrição
`coony/`	Configurações principais (`settings.py`, `urls.py`, `wsgi.py`, `asgi.py`)
`usuarios/`	App Django com models, forms, views, urls, templates, static e migrações
`usuarios/templates/usuarios/`	Páginas desktop/mobile, dashboard, social, perfil, chat, notificações, componentes (sidebar, navbar, toast)
`usuarios/static/`	CSS de cada página, scripts (`script/chat.js`, `script/app.js`), assets (fonts, icons, imgs)
`media/`	Uploads (fotos de usuários em `usuarios/fotos/`, posts, capas de eventos)
`db_viewer.py`	Ferramenta desktop para explorar o SQLite local com login ADMIN/ADMIN
`requirements.txt`	Dependências de produção/desenvolvimento
`TOAST_IMPLEMENTATION.md` / `TOAST_SUMMARY.txt`	Documentação do sistema de notificações

Configuração & Ambiente

- Variáveis suportadas (via `.env.*`): `DJANGO_SECRET_KEY`, `DJANGO_DEBUG`, `DJANGO_ALLOWED_HOSTS`, `DATABASE_URL`.
- `STATICFILES_DIRS` aponta para `usuarios/static`; `STATIC_ROOT` resolve em `staticfiles/` para `collectstatic`.
- Uploads são servidos de `media/` quando `DEBUG=True`; `MEDIA_URL=/media/`.

- `django_user_agents.middleware.UserAgentMiddleware` habilita `request.user_agent` nos templates/views para alternar UI.

Modelos de Dados (`usuarios/models.py`)

Model	Campos-chave / Função
`Usuario`	Perfil custom (nome, telefone, email único, senha hash, localização, modalidades, bio, foto, slug `username`). Métodos para hash/checagem e geração de handle único.
`Post`	Conteúdo social com texto, imagem opcional, localização e relação `likes` (ManyToMany com `Usuario`).
`PostLikeEvent`	Histórico de curtidas para construir notificações; garante unicidade por post/usuário.
`Comment`	Comentários ligados a `Post` com ordering cronológico.
`Conversation`	Conversas privadas (WhatsApp-like) entre dois usuários; gera `conversation_key` e mantém `updated_at`.
`Message`	Mensagens com flags de leitura, exclusões seletivas/global, autor, timestamps.
`Evento`	Eventos criados pelo usuário com mídia (capa + até 3 imagens), modalidade, data/hora/local, limite de participantes e `favorited_by` (ManyToMany para favoritos persistentes).

Forms Principais (`usuarios/forms.py`)

- `RegistrationForm` / `LoginForm` para onboarding.
- `PostForm` (texto, imagem, localização) e `CommentForm` para o feed social.
- `EventoForm` com validação de imagem obrigatória e widgets específicos (date/time inputs, placeholders, accept image/*).

Autenticação e Helpers

- Sessões utilizam a chave `session['usuario_id']` em vez de `request.user`.
- `_get_logged_user` e `_json_auth_required` encapsulam a recuperação/validação do usuário em views normais e APIs JSON.
- Senhas são armazenadas usando `django.contrib.auth.hashers` (mesmo sem `AbstractUser`).

Rotas & Views

Público / Onboarding

Rota	Método	View	Descrição
`/`	GET	`index`	Landing desktop com formulários de login/registro + toasts
`/mobile/`	GET	`mobile`	Variante mobile da tela inicial
`/register/`	POST	`register_view`	Cria `Usuario`, faz login e redireciona ao dashboard

Rota	Método	View	Descrição
`/login/`	POST	`login_view`	Aceita e-mail, nome ou `@username`; utiliza `normalize_username`
`/logout/`	GET	`logout_view`	Limpa sessão e mostra toast de despedida

Dashboard & Eventos

Rota	Método	View	Descrição
`/dashboard/`	GET	`dashboard`	Lista eventos do usuário (desktop); redireciona mobile
`/dashboard/mobile/`	GET	`dashboard_mobile`	Layout responsivo com cards empilhados
`/eventos/criar/`	GET/POST	`create_event`	Formulário + listagem "Meus eventos" com galeria; alterna sidebar/navbar por largura
`/eventos/toggle_favorite/<id>/`	POST	`toggle_favorite_event`	Endpoint AJAX usado por ambos dashboards para favoritar

Social & Engajamento

Rota	Método	View	Descrição
`/social/`	GET/POST	`social`	Feed com criação de posts, upload de imagem, contagem de likes/comentários, remoção do próprio post
`/social/like/<id>/`	POST	`like_post`	Alterna curtida e gera `PostLikeEvent`
`/social/comment/<id>/`	POST	`comment_post`	Adiciona comentário via `CommentForm`
`/social/delete/<id>/`	POST	`delete_post`	Autores removem seus posts; coberto por testes
`/notifications/`	GET	`notifications`	Consolida mensagens recentes, likes e comentários com CTA (chat/social)

Perfil & Utilidades

Rota	Método	View	Descrição
`/perfil/`	GET/POST	`perfil`	Atualiza dados, @handle, email, localização, modalidades dinamicamente, bio e foto com CropperJS/cortes

Rota	Método	View	Descrição
`/chat/`	GET	`chat`	SPA-like para mensagens privadas; injeta dados atuais e URLs das APIs

APIs de Chat (JSON)

- `GET /chat/api/conversations/` → lista conversas (inclui preview, último timestamp).
- `GET /chat/api/search/?q=` → busca usuários por nome/@, exclui o atual.
- `POST /chat/api/conversations/start/` → inicia conversa privada via payload JSON.
- `GET /chat/api/conversations/<id>/messages/` → mensagens filtradas para o usuário atual.
- `POST /chat/api/conversations/<id>/messages/send/` → envia texto.
- `POST /chat/api/messages/<id>/delete/` → remove para si ou todos com regras de permissão.

Páginas e Componentes de Interface

- **`usuarios/index.html` / `mobile.html`:** telas de login/registro com includes `toast.html` + `messages.html`.
- **Dashboard desktop (`dashboard.html`):** cards com imagem, descrição truncada, informações (local/data) e botão. Inclui busca fictícia, filtros estáticos e toggles de favoritos por fetch + `csrftoken`.
- **Dashboard mobile:** mesma fonte de dados com layout comprimido, header com ícones, e fallback quando não há eventos.
- **`create_event.html`:** layout com hero, formulário responsivo, instruções de mídia e listagem rica dos eventos do criador (covers + gallery thumbnails).
- **`social.html`:** feed completo (tabs, share box, preview de imagem, modal de localização, contadores, botões de curtir/comentar/compartilhar) com JS para navegação mobile/desktop.
- **`perfil.html`:** formulário multi-seção com tags de modalidades, sincronização de inputs ocultos, upload/crop/remoção de foto e navegação responsiva.
- **`notifications.html`:** cards com timeline relativa usando `timesince`. (Estrutura definida na view.)
- **`chat.html`:** layout de duas colunas (lista de contatos + corpo do chat), integra `emoji-picker`, ações (call/voltar), e carrega `static/script/chat.js` para consumir as APIs.
- **Componentes compartilhados:** `sidebar.html`, `navbar_mobile.html`, `navbar.html`, `toast.html`, `messages.html`, `components/messages.html` para bridging com Django messages.

Assets Estáticos & JS

- CSS específico por página em `usuarios/static/css/` (`home.css`, `home(mobile).css`, `create-event.css`, `perfil.css`, `chat.css`, etc.).
- Scripts:
  - `static/script/app.js` e `static/script/chat.js` para comportamento de navegação, chat e interações.
  - Inline scripts nos templates para toggles (favoritos, navbars, pre-visualização de imagens) e modais.
- Fonts/ícones: Google Fonts, Material Symbols (com `font-variation-settings` para hearts preenchidos), Boxicons, Remix Icons, CropperJS, emoji-picker.
- Toast demo disponível em `usuarios/static/toast-demo.html`.

Sistema de Toasts

- Localizado em `usuarios/templates/usuarios/components/toast.html` e descrito em `TOAST_DOCUMENTATION.md`.
- Tipos suportados: success/error/warning/info com animações CSS, barra de progresso e API JS global `Toast.*`.
- Django `messages` adiciona `extra_tags='toast'` para exibir automaticamente na próxima resposta.

Mídia e Uploads

- Uploads de usuários: `usuarios/fotos/`.
- Postagens: `media/posts/`.
- Eventos: `media/eventos/capa/` e `media/eventos/detalhes/`.
- Funções de profile permitem remover/resetar foto e usar fallback `static/img/default-avatar.svg`.

Ferramentas e Utilidades

- `db_viewer.py`: interface Tkinter com login ADMIN/ADMIN para listar tabelas, filtrar, exportar CSV, CRUD básico e visualizar imagens (via Pillow). Útil para inspeção sem acessar admin Django.
- `db_viewer_README.md`: instruções rápidas para o viewer.

Testes Automatizados (`usuarios/tests.py`)

- `DeletePostViewTests` garante que apenas o autor remove postagens.
- `UsernameGenerationTests` valida criação automática de usernames únicos.
- `LoginWithUsernameTests` cobre login via `@handle` com/sem arroba.

Executando Localmente

```
python -m venv .venv
.\.venv\Scripts\Activate.ps1
pip install -r requirements.txt

# Configure variáveis opcionais
$env:DJANGO_DEBUG = 'True'
$env:DJANGO_ALLOWED_HOSTS = 'localhost,127.0.0.1'

python manage.py makemigrations usuarios
python manage.py migrate
python manage.py runserver
```

- Acesse `http://localhost:8000/`. Para usar recursos que dependem de upload (eventos, perfil), garanta permissões de escrita na pasta `media/`.
- `python manage.py createsuperuser` é opcional se desejar usar `/admin/` (apenas `Usuario` está registrado por padrão).

Considerações de Deploy

- Defina `DJANGO_SECRET_KEY` forte e `DJANGO_DEBUG=False`.

- Popule `DJANGO_ALLOWED_HOSTS` com domínio(s) reais; `CSRF_TRUSTED_ORIGINS` é derivado automaticamente.
- Configure `DATABASE_URL` (PostgreSQL recomendado em produção).
- Execute `python manage.py collectstatic` para enviar CSS/JS/imagens para `STATIC_ROOT`.
- Use storage dedicado para `MEDIA_ROOT` (S3, Azure Blob, etc.) ou monte volume específico.

Próximos Passos Possíveis

1. Expor APIs REST para eventos/posts (Django REST Framework).
2. Migrar `Usuario` para um `AbstractUser` custom se for necessário integrar com Django auth padrão.
3. Consolidar scripts inline em bundles versionados (`static/script/`).
4. Adicionar testes para chat APIs e favoritos.
5. Criar componentes reusáveis para cards de eventos e feed, facilitando evolução futura.