**Project: Encrypted Social Media App (Django + Channels)**

Author: Tausif  
Date: October 15, 2025  
Target Environment: Windows 11, VS Code, Python 3.11+

# 1) Repository Structure (Monolith - Django)

app\_root/  
├─ .env.example  
├─ manage.py  
├─ requirements.txt  
├─ README.md  
├─ Procfile (for deployment)  
├─ Dockerfile (optional)  
├─ tailwind.config.js (if using Tailwind CLI or CDN in templates)  
├─ package.json (optional, if using Tailwind build)  
├─ compose.yaml (optional; postgres + redis + web)  
├─ config/ (Django project)  
│ ├─ \_\_init\_\_.py  
│ ├─ asgi.py (Channels entrypoint)  
│ ├─ settings.py  
│ ├─ urls.py  
│ └─ wsgi.py  
├─ crypto\_core/ (encryption + key management)  
│ ├─ \_\_init\_\_.py  
│ ├─ keys.py (MASTER\_KEY handling, envelope encryption)  
│ ├─ fields.py (EncryptedTextField, helpers)  
│ └─ utils.py (base64, AEAD helpers, rotation hooks)  
├─ users/ (auth + profiles)  
│ ├─ \_\_init\_\_.py  
│ ├─ admin.py  
│ ├─ apps.py  
│ ├─ forms.py (Register/Login)  
│ ├─ models.py (Profile with key\_blob, encrypted fields)  
│ ├─ urls.py  
│ ├─ views.py  
│ ├─ serializers.py (if DRF is used)  
│ ├─ templates/users/ (login.html, register.html, profile.html)  
│ └─ tests/  
├─ posts/ (basic posting)  
│ ├─ models.py (Post.enc\_body, author)  
│ ├─ views.py (CRUD)  
│ ├─ urls.py  
│ ├─ templates/posts/ (list.html, create.html, detail.html)  
│ ├─ serializers.py (optional)  
│ └─ tests/  
├─ messaging/ (real-time DM with self-destruct)  
│ ├─ consumers.py (WebSocket handlers)  
│ ├─ routing.py (URLRouter for ws)  
│ ├─ models.py (Thread, Message: enc\_body, seen\_at, delete\_after)  
│ ├─ views.py (REST endpoints for threads/messages if needed)  
│ ├─ urls.py  
│ ├─ tasks.py (schedule delete after seen)  
│ ├─ templates/messaging/ (inbox.html, thread.html)  
│ └─ tests/  
├─ static/ (css/js/images)  
│ ├─ css/  
│ └─ js/  
└─ templates/ (base layout)

# 2) Dependencies

* Django 5.x
* Django REST Framework (optional but recommended)
* Django Channels + channels-redis
* Redis (server)
* cryptography (AEAD via Fernet/AES-GCM)
* Celery (or django-q2) + Redis as broker (deletion scheduling)
* psycopg2-binary (PostgreSQL driver for production)
* Tailwind CSS (+ DaisyUI) for UI

# 3) Environment Variables (.env)

DJANGO\_SECRET\_KEY="..."  
DJANGO\_DEBUG="1/0"  
DATABASE\_URL="postgres://user:pass@host:5432/db"  
REDIS\_URL="redis://host:6379/0"  
DJANGO\_MASTER\_KEY="base64-32B-key"  
ALLOWED\_HOSTS="localhost,127.0.0.1,example.com"

# 4) Security Design (meets PDF constraints)

* Passwords: Django auth with Argon2PasswordHasher (salted + strong KDF).
* Field Encryption: per-user USER\_DATA\_KEY (random 32B). Stored as key\_blob encrypted under MASTER\_KEY (envelope encryption).
* AEAD: Encrypt all sensitive fields (posts, messages, profile) with USER\_DATA\_KEY; integrity via MAC/tag.
* Decryption path: Only after auth + permission checks on the server; never log plaintext.
* Key Rotation: rewrap key\_blobs with rotated MASTER\_KEY, optional per-user re-encrypt.
* TLS on deploy, secure cookies, CSRF, HSTS.

# 5) Realtime Messaging & Self-Destruct Flow

1. WebSocket endpoint: /ws/threads/{thread\_id}/ via Channels.
2. Send: validate perms, encrypt message body, save, broadcast.
3. Seen: client emits “seen”; server sets seen\_at=now() and schedules delete at seen\_at+5min.
4. Deletion: Celery ETA task + periodic safety sweep; broadcast “message\_deleted” over WS.
5. UX: countdown badge appears on seen messages (5:00 → 0:00), then vanish without reload.

# 6) HTTP/WS Endpoints (sketch)

POST /api/auth/register  
POST /api/auth/login  
GET /api/posts  
POST /api/posts  
GET /api/posts/{id}  
GET /api/threads  
POST /api/threads  
GET /api/threads/{id}/messages  
POST /api/threads/{id}/messages  
POST /api/messages/{id}/seen (triggers delete schedule)  
WS /ws/threads/{id}/ (events: message\_new, message\_seen, message\_deleted)

# 7) Data Models (minimal)

User (built-in)  
Profile(user OneToOne, key\_blob BLOB, enc\_full\_name TEXT, enc\_email\_copy TEXT, ...)  
Post(author FK, enc\_body TEXT, created\_at)  
MessageThread(participants M2M)  
Message(thread FK, sender FK, enc\_body TEXT, created\_at, seen\_at NULLABLE, delete\_after NULLABLE)

# 8) UI/UX Plan

* Base layout: navbar (brand, search, profile menu), container with responsive grid.
* Feed: card list of posts (author, timestamp, body), create-post modal.
* Messaging: split pane (threads list + chat area), sticky composer, sent/delivered/seen ticks, countdown chip post-seen.
* Theme: Tailwind + DaisyUI (light/dark), mobile-friendly.

# 9) Implementation Steps

1. Scaffold project, apps (users, posts, messaging), settings for Channels + Redis.
2. Users: registration/login, Argon2, Profile with key\_blob, encrypt profile fields.
3. Crypto core: MASTER\_KEY read, per-user key generation, EncryptedTextField, helpers.
4. Posts: encrypted body, CRUD views/templates.
5. Messaging: models, WS consumers, routing, permissions, broadcast on send.
6. Seen flow: endpoint + WS event; set delete\_after; schedule Celery ETA task.
7. Safety sweep periodic task; WS “message\_deleted” event.
8. UI polish: Tailwind/DaisyUI components, countdown badge in chat.
9. Tests: unit (crypto, models), integration (WS), E2E happy paths.
10. Deployment: Postgres + Redis, env vars, ASGI server, worker process.

# 10) Testing & Acceptance Criteria

* Passwords stored hashed+salted; login succeeds with correct creds.
* All defined sensitive fields stored as ciphertext in DB.
* Messages appear in real-time without page reload.
* When recipient triggers “seen”, a countdown starts; message is deleted server-side after 5 minutes.
* Safety sweep deletes overdue messages even if worker was down.
* No plaintext user data in logs.
* Deploy: site loads over HTTPS; WebSocket upgrades succeed; env keys present.

# 11) Deliverables

* Full Django source with instructions.
* .env.example with required variables.
* Screens: Login, Register, Feed, Inbox, Thread.
* Deployment guide (Render/Railway/Fly.io).