# Cloakpost – Full Frontend Development Context for Next Phase

Date: October 16, 2025

## Purpose

This document provides complete context for continuing the Cloakpost project into its next phase — building the \*\*entire frontend\*\* (from user authentication to encrypted messaging) using Django templates, Tailwind CSS, and real-time WebSocket updates (no page refresh). It is meant to be uploaded into a new chat so the assistant knows the full backend status and what to build next.

## Backend Status

✅ The backend is fully complete and passes all test cases. It satisfies every requirement outlined in the CSE447 Lab project PDF, including encryption, authentication, key management, and auto-deletion logic.

Key components implemented:

* User registration/login (PBKDF2 password hashing).
* AES-256-GCM message encryption via crypto\_core app.
* Encrypted messaging with Threads and Messages (server-side AES encryption).
* Django Channels WebSocket consumer for real-time communication.
* Seen → 5-minute auto-deletion via Celery ETA + periodic sweep.
* Complete test coverage for users, posts, messaging, and Celery deletion flow.

## Next Phase: Full Frontend Scope

The next task is to build the full UI workflow — login to messaging — entirely in Django templates.

* Required features:
* ✅ Login, Register, Logout pages using Django auth views (styled with Tailwind + DaisyUI).
* ✅ Posts feed and basic post creation view.
* ✅ Threads list page showing all conversations for the logged-in user.
* ✅ Thread detail page with real-time chat interface (WebSocket).
* ✅ Chat interface must support instant message send/receive (no page reload).
* ✅ Show 'Seen' indicator and 5-minute countdown badge for messages after being read.
* ✅ Messages auto-remove from the DOM when deleted (after Celery-triggered deletion).

## Frontend Stack

Use Django’s built-in template engine with Tailwind CSS and DaisyUI. Include a small vanilla JavaScript client (`static/js/chat.js`) for real-time WebSocket communication. Keep everything server-rendered for simplicity and compatibility with Django’s auth/session management.

## Files to Implement

* templates/base.html – shared layout (Navbar, flash messages, etc.).
* templates/users/login.html – login form.
* templates/users/register.html – registration form.
* templates/posts/feed.html – posts list & composer.
* templates/messaging/threads.html – list of message threads.
* templates/messaging/thread.html – real-time chat UI (connected to WebSocket).
* static/js/chat.js – WebSocket client logic (send, receive, seen, countdown, delete).

## Behavior Expectations

The frontend must allow smooth navigation without full page reloads between chats. All real-time updates (new message, seen, deleted) should appear instantly through WebSocket events.

* Event flow:
* Client sends { action: 'send', body: 'text' } → message\_new broadcast.
* Client receives message\_new → append to chat instantly.
* Client sends { action: 'seen', message\_id: id } → countdown starts.
* Server broadcasts message\_seen → other clients show countdown badge.
* Server broadcasts message\_deleted (after 5min) → remove message DOM element.

## Deployment Context

The backend is ready for deployment to Render or Railway with Redis + PostgreSQL. Frontend additions will not affect production settings. Once the frontend is complete, a deployment guide will be added with environment variables and process setup for Daphne + Celery workers.

## Next Chat Instructions

When this document is uploaded in a new chat, the assistant should immediately start generating the full frontend code (Django templates, Tailwind integration, and chat.js WebSocket logic). The UI must be functional and visually clean.