Advanced International Journal for Research (AIJFR)
E-ISSN: 3048-7641 ● Website: www.aijfr.com ● Email: editor@aijfr.com

On-Chain Social Network D-App

Prof. Nishant Dhage[1], Anukuma Chauhan[2], Akansha Pranjale[3], Bhairavi Kachave[4], Muskan Gedam[5], Sakshi Mandalwar[6]

[1] Assistant Professor, Computer Science & Engineering, Priyadarshini Bhagwati College of Engineering
[2,3,4,5,6] B.Tech Students, Computer Science & Engineering, Priyadarshini Bhagwati College of Engineering

Abstract

This paper outlines the design and implementation of a decentralized social application built on the Internet Computer (ICP) platform, featuring a Rust-based canister backend and a TypeScript/JavaScript frontend fully hosted on-chain. It emphasizes complete decentralization, type-safe communication through Candid, and a unified DFX-driven workflow for both development and deployment. The study explores integration with Internet Identity for secure authentication, models social graph data to support user interactions, and applies performance optimization techniques for a responsive user experience. It also discusses security measures consistent with ICP's unique execution and web-serving architecture. Positioned within the broader field of decentralized online social networks (DOSNs), this work demonstrates how ICP's vertically integrated stack enables end-to-end decentralized applications with scalable, secure, and transparent infrastructure—advancing research and practice in blockchain-based social systems and highlighting ICP's potential as a foundation for next-generation, user-controlled digital platforms.

Keywords: Internet Computer, canister smart contracts, Candid, decentralized social network, Internet Identity, Web Authn, asset canister, stable memory, DFX, on-chain web serving.

1. Introduction

Decentralized social media demands a platform capable of hosting logic, data, and web assets directly on-chain, which ICP provides through canisters that serve both compute and frontend without external servers. This paper outlines a practical ICP full-stack pattern for a social dapp, pairing a Rust canister with a typed JavaScript client over Candid, enabling rapid iteration locally and seamless deployment to the public Internet Computer.

2.Background and Related Work

Efforts in DOSNs highlight challenges in privacy, moderation, and interoperability, where architectures often rely on EVM + IPFS + gateways, complicating UX and trust surfaces. ICP's integrated compute,

storage, and web serving offers a single-trust domain for interactive apps, demonstrated by ecosystem social dapps and official guidance on canisters, Candid, and Internet Identity.

2.1 Internet Computer Overview

ICP executes Web Assembly canisters across subnets, providing deterministic state, stable storage across upgrades, and native web asset serving from the chain, removing dependence on centralized clouds. Developers interact via DFX, define interfaces in Candid, and can integrate privacy-preserving authentication through Internet Identity using Web Authn-based device credentials.

2.2 DOSN Patterns and ICP Fit

Typical DOSN (Decentralized Online Social Network) stacks distribute compute and storage across heterogeneous systems, whereas ICP collapses this into canisters that handle business logic, persistence, and UI delivery, simplifying latency and deployment. This consolidation enables verifiable execution, reduces operational complexity, and supports professional-grade UX akin to centralized platforms while maintaining user control.

3. Problem Statement and Objectives

Traditional centralized social media platforms concentrate authority over user identities, data storage, and content moderation. This centralization results in significant privacy risks, limited transparency, user dependency on platform providers, and the inability to freely migrate or control personal data. These issues create a need for decentralized alternatives that empower users with full ownership and transparency.

The objective of this work is to design and implement a full-stack on-chain decentralized social application leveraging the Internet Computer (ICP). The system aims to restore user control while ensuring a responsive user experience (UX) and efficient developer workflows. The key goals include establishing a reproducible development workflow, defining a type-safe canister interface, integrating Internet Identity for secure authentication, and developing a scalable architecture that supports future governance mechanisms and interoperability within the decentralized ecosystem.

4. Used Languages

The implementation uses Rust for the canister backend to leverage memory safety and performance, TypeScript/JavaScript for typed client bindings and UI logic, CSS for styling, and HTML for markup, reflecting a typical ICP full-stack composition for on-chain web apps.

4.1 Languages and Structure

The project pairs a Rust canister create exposing Candid methods for posts, profiles, and follow relationships with a web client that consumes autogenerated declarations for type-safe calls, served by an asset canister or local dev server during iteration. This separation preserves strong typing across the boundary and supports upgrade-safe evolution of the interface.

4.2 Tooling and Scripts

DFX manages local replica, build, and deploy cycles; scripts generate Candid declarations for the frontend to ensure compatibility with the canister interface, and a dev server proxies API requests to the local replica for fast reloads. Environment configuration prevents fetching the root key in production when hosting outside the DFX pipeline by setting the target network appropriately or overriding declarations.

5. Development Workflow

Local development begins with launching a replica, deploying canisters, generating type declarations from Candid, and running a frontend dev server that proxies to the replica, enabling rapid feedback while preserving the on-chain interface contract. Production deployment serves the compiled frontend via the asset canister and points the agent to the IC network with proper root key handling.

5.1 Local Replica and Deployment

A local replica runs in the background for fast, deterministic testing; deploying builds the Rust canister, installs it, and publishes the asset canister, making the app reachable at the local replica endpoint with the asset canister ID. Iteration cycles consist of code changes, redeploys, and hot-reload for frontend while backend changes regenerate the interface.

5.2 Candid Generation and Declarations

Generating declarations from the canister's Candid ensures the frontend has up-to-date, typed method signatures, minimizing integration errors and enabling compile-time checks for interface drift. This step is recommended before launching the dev server and is often automated during deploy to keep interfaces synchronized.

5.3 Frontend Dev Server and Proxying

A local dev server runs on a standard port and proxies canister API calls to the replica port, allowing modern frontend tooling and HMR while preserving real canister interactions behind the proxy. This setup decouples UI iteration speed from canister rebuilds when only client changes are made.

6. System Architecture

The architecture centers on a Rust canister implementing social primitives and an asset canister hosting the UI, with typed communication via Candid-generated bindings and agent actors in the client. In production, the entire stack—logic, data, and web—is served on-chain, with identity provided by Internet Identity and canister upgrades preserving state via stable memory patterns.

6.1 Rust Canister Backend

Backend methods expose create/read operations for profiles, posts, and follow edges, using update calls for mutations and query calls for reads to balance consistency and performance. Stable memory and

schema versioning preserve user data across upgrades, and access control leverages caller principals to authorize actions.

6.2 Asset Canister Frontend

UI assets are uploaded to and served by an asset canister so end users retrieve the application directly from the chain, ensuring integrity and minimizing centralized hosting dependencies. The UI constructs an agent and actors using the generated declarations to call canister methods with typed safety.

6.3 Typed Actor Bindings

Autogenerated declarations from Candid define the client-side actors, aligning types between Rust and TypeScript, reducing runtime errors, and documenting the public API contract as part of the build pipeline.

7. Identity and Authentication

Internet Identity provides passwordless, device-bound authentication with WebAuthn, minimizing correlatable identifiers and aligning with privacy-first social design [2]. The client obtains an II delegation and configures the agent to sign requests, enabling authenticated canister calls without traditional username/password flows.

7.1 Internet Identity Integration

Integration involves adding the II canister as an identity provider, initiating WebAuthn flows in the client, and persisting delegation for session continuity, with recovery options for device loss. This approach reduces phishing risk and removes email/phone from the identity plane by default.

7.2 Agent Configuration and createActor

A custom actor constructor wires the authenticated agent, target canister ID, and network configuration, ensuring correct root key usage and secure request signing across environments. This layer abstracts II integration and enables future multi-identity or wallet support if required.

8. Data Model and Ownership

All core social data lives within canisters under protocol rules, providing verifiable handling and enabling user-centric governance models over time. Read-heavy endpoints are optimized as queries, while writes are transactional updates; privacy-sensitive fields rely on principal-based access checks in canister code.

8.1 Profiles, Posts, Social Graph (Design)

Profiles map principal IDs to metadata; posts store author, timestamp, content, and references; follow edges encode directed relationships for feed construction and discovery. Pagination and indexing structures are introduced to support scalable listing and timeline queries without exhausting canister cycles.

8.2 Stable Memory and Upgrades

Persistent data is stored in stable memory with versioned schemas and migration routines executed during canister upgrades to ensure continuity and forward compatibility. Careful serialization formats and bounded allocations protect against upgrade failures and memory fragmentation.

9. Performance and UX

Perceived latency is improved through client-side caching, optimistic UI for post/like actions, and batched queries, while on the backend, separating read queries from updates increases responsiveness. Serving the UI on-chain and minimizing cross-domain hops reduces cold-start overhead and improves consistency across clients.

9.1 Client Caching and Pagination (Plan)

Cursor-based pagination limits payload size and supports infinite scroll, with cache invalidation on write acknowledgments to keep timelines fresh without overfetching. Structured query methods return compact summaries with lazy expansion for detail views to reduce cycles and bandwidth.

9.2 Canister Partitioning and Scaling

Sharding by feature (profiles, posts, graph) or by keyspace range distributes load across canisters and subnets, with cross-canister calls encapsulated behind stable interfaces to maintain modularity. Background tasks can precompute popular timelines or indexes within cycle budgets to accelerate common reads.

10. Security Considerations

Production builds must avoid dev-only root key fetching and ensure the agent targets the IC network with validated certificates, while environment variables in generated declarations are pinned for safety. Input validation, rate limits, and role-based controls are enforced in canister methods to mitigate spam, abuse, and unauthorized mutations.

10.1 Root Key Handling in Production

When hosting the frontend outside the DFX runtime, environment overrides ensure the agent does not fetch a local root key and instead trusts IC’s certificate chain, preventing man-in-the-middle development configurations from leaking to production. Build-time replacement of network constants aligns declarations with the target network.

10.2 Authorization, Rate Limits, Moderation

Authorization binds actions to caller principals; rate-limiting policies throttle abusive patterns; and moderation introduces report workflows and admin or community actions consistent with DOSN safety expectations. Logging and audit trails enable post-incident analysis while respecting privacy boundaries inherent to decentralized identity.

11. Comparative Context

ICP's unified canister model contrasts with architectures reliant on off-chain servers or IPFS gateways, reducing trust assumptions and simplifying developer operations for interactive social UX. Native web serving and Internet Identity integration further differentiate ICP for DOSNs seeking strong UX without central dependencies.

11.1 ICP vs EVM+IPFS Stacks

While EVM+IPFS can decentralize storage, dynamic interactivity often reintroduces centralized middleware; ICP executes logic and serves UI directly on-chain, preserving the decentralization envelope for both data and app surface. This reduces architectural sprawl and latency associated with multi-stack compositions.

11.2 Path to Production DOSN

A production path builds from the described scaffold into feature-complete profiles, posts, and graph services, adds II authentication, enforces access control, and iteratively scales via canister partitioning and pagination while serving the UI on-chain. Governance hooks and interoperability can then be layered without architectural rewrites due to the stable, typed canister interfaces.

12. Roadmap

Short-term goals include defining Candid methods for core social entities, integrating Internet Identity, and establishing pagination and caching in the client. Medium-term work partitions canisters, adds moderation and media handling, and hardens upgrade paths; long-term plans address DAO governance and interop standards.

12.1 Short-term Features

Implement create/read endpoints for profiles, posts, and follows; add II login; set up cursor pagination; and ensure stable memory layouts with migration scaffolding and tests for upgrades.

12.2 Medium-term Architecture

Separate canisters by domain, introduce content reporting and admin workflows, support chunked media storage, and add background indexing within cycle constraints to accelerate common queries.

12.3 Long-term Governance and Interop

Evolve toward community governance over policies and features, and explore cross-network identity or content bridges aligned with DOSN standardization efforts without compromising on-chain integrity.

13. Limitations and Open Challenges

The base implementation is minimal and requires substantial feature work, tests, and robust schemas to reach production readiness, especially for media and large-scale feeds. Moderation at scale, cost management for storage and cycles, and interoperable identity remain active areas needing careful mechanism design.

14. Conclusion

A full-stack ICP design using a Rust canister backend, on-chain frontend, Candid-typed interfaces, and Internet Identity enables a decentralized social application with verifiable execution and strong UX foundations. The outlined workflow and architecture provide a practical path from prototype to production while preserving user control and minimizing centralized dependencies.

Acknowledgments

Appreciation is extended to contributors to the development workflow, Candid generation patterns, and environment configuration that streamline iterative canister development and safe production deployment.

Reference

1. Navigating Decentralized Online Social Networks: “An Overview of Technical and Societal Challenges in Architectural Choices” Ujun Jeong, Lynnette Hui Xian Ng, Kathleen M. Carley, Huan Liu (2025).
2. DeSocial: “ Blockchain-based Decentralized Social Networks “Jingyuan Huang, Xi Zhu, Minghao Guo, Yongfeng Zhang (2025).
3. “Architecture for Protecting Data Privacy in Decentralized Social Networks” Quang Cao, Katerina Vgena, Aikaterini-Georgia Mavroeidi, Christos Kalloniatis, Xun Yi, Son Hoang Dau (2024).
4. Bluesky and the AT Protocol: “Usable Decentralized Social Media” Martin Kleppmann, Paul Frazee, Jake Gold, Jay Graber, Daniel Holmgren, Devin Ivy, Jeromy Johnson, Bryan Newbold, Jaz Volpert (2024).
5. “Decentralised Social Media” Konstantinos Votis, Ioannis Revolidis, Joshua Ellul, Catarina Ferreira da Silva, Daniel Szegö, Amit Joshi (2023).
6. DFINITY Team, **“The Internet Computer for Geeks”** (Internet Computer / DFINITY whitepaper), 2023. core technical whitepaper describing canisters, chain-key tech, and web serving.