REQUEST HYDRATION & CSR

SSR-INITIAL PAINT

Execution begins with Svelte Kit Server-Side Rendering. Campus metadata is pre-serialized, ensuring a Time to Interactive (TTI) < 200ms.

SPA TAKEOVER

Client-side hydration enables a highly reactive SPA. Assets are distributed globally via Vercel Edge for millisecond response times.

SERVICE WORKER

Background sync engine manages offline persistence and local cache invalidation, ensuring instant loads on poor campus Wi-Fi.

STATE RECONCILIATION LOGIC

OPTIMISTIC UI

CLIENT-SIDE UPDATE

The UI store is updated instantly on user action. Background workers coordinate REST mutations with the Express backend.

```
store.update(state => ({ ...state, posts: [newPost, ...state.posts] }));
```

SYNC RECONCILED

ATOMIC ROLLBACK

The system compares REST responses against local state. If a conflict occurs (409), the UI store rollbacks to last-known-good state.

```
if (res.status === 409) {
  store.set(res.latest_state); notify_conflict();
}
```

ZERO-I/O MEDIA STRATEGY

🔑 HMAC-Signed URL Pipeline

The API server issues a temporary cryptographic signature with a 15-minute TTL, allowing secure client-side resource management.

⬆️ Direct-to-Storage

The user's device performs an authenticated PUT directly to Cloudinary/S3. This architectural decision bypasses the API to preserve compute bandwidth.

🗄️ Resource Neutrality

The database only persists the resulting public CID/Hash, ensuring our logic layer remains purely stateless and media-neutral.

💲 Cost Efficiency

By offloading heavy I/O tasks, we achieve significant savings on server-side egress costs and drastically reduce upload failure rates.

DATABASE SCHEMA ENTITIES

Entity	Primary Key	Type	Security Policy
Identity (Users)	id	UUIDv4	Owner-Only Write
Campus Graph	init_id, trgt_id	Composite	Reciprocal Read
Messaging	msg_id	BigInt	Peer-to-Peer Isolation
Social Feed	post_id	UUIDv4	Tenant-Only Read

* Postgres constraints enforce referential integrity across all partitioned tables.

PARTITION-READY ISOLATION

COMPOSITE B-TREE

Indexes are created on (university_id, created_at), ensuring $O(\log N)$ lookups within tenant partitions and avoiding global table scans.

CASCADE INTEGRITY

Foreign Keys are configured with cascading deletes. Account removal triggers an atomic cleanup of posts and messaging history.

TENANT ISOLATION

Isolation is enforced at the query layer. Every mutation/selection includes a university_id predicate to prevent accidental leakage.

AUTH & DATABASE GUARDRAILS

IDENTITY

STATELESS JWT AUTH

Authentication is strictly stateless. JWTs contain custom claims for uni_id, preventing cross-campus access even if an endpoint is spoofed.

```
{ "sub": "user_123", "uni": "campus_456", "role": "student" }
```

PERSISTENCE

POSTGRES RLS POLICIES

Row-Level Security is our fail-safe. The DB engine examines session identity and physically refuses to release rows if ownership fails.

```
CREATE POLICY tenant_isolation ON posts
USING ( uni_id = current_setting('uni_id') );
```

NETWORK EDGE PROTECTION

WAF FILTERING

Global Vercel/Cloudflare WAF rules automatically block malicious IPs and known botnet signatures before reaching compute.

SQLI DETECTION

Prisma ORM automatically parameterized all queries, mathematically eliminating SQL Injection vectors at the model layer.

RATE LIMITING

Strict burst-rate limiting (100 req/min) preserves serverless compute resources for legitimate student traffic.

REAL-TIME EVENT PIPELINE (SOCKET.IO)

Persistent Bidirectional WebSockets

The platform maintains persistent bidirectional WebSocket connections via Socket.io alongside REST. This dual-channel architecture eliminates polling overhead entirely, removing redundant HTTP round-trips from the critical path.

Sub-100ms Real-Time Chat

The Socket.io event pipeline delivers sub-100ms real-time chat latency. Messages are emitted, acknowledged, and rendered in the recipient's UI store before a traditional REST cycle would even complete its handshake.

Instant Campus Feed Updates

Campus feed updates are broadcast instantly to all connected clients within a tenant namespace. New posts, reactions, and comments propagate in real-time without requiring manual refresh or background polling intervals.

PROGRESSIVE SCALABILITY (MONOLITH → MICROSERVICES)

PHASE 1 — HACKATHON MVP

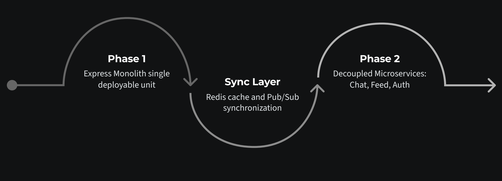
Fast Express Monolith

The platform launches as a fast Express "Monolith" optimized for the Hackathon MVP. All compute — Chat, Feed, Auth, and Media — runs within a single deployable unit for maximum development velocity and minimal operational complexity.

PHASE 2 — PRODUCTION SCALE

Decoupled Microservices

The backend is architecturally designed to decouple into highly scalable Microservices. Chat compute is separated from Feed compute, each independently scalable. A **Redis Cache and Pub/Sub layer** synchronizes state across services, ensuring eventual consistency and horizontal elasticity under campus-scale load.



This progressive architecture ensures we ship fast for the MVP while maintaining a clear, pre-engineered migration path to production-grade microservices without rewriting core business logic.

STRICT MONOREPO ARCHITECTURE

NPM Workspaces / Turborepo

The system is cleanly segmented into a Monorepo using NPM Workspaces and Turborepo. All packages, dependencies, and build pipelines are managed from a single repository root with deterministic resolution.

@Clink_Frontend & @Clink_Backend

Distinct @Clink_Frontend and @Clink_Backend namespaces enforce strict separation of concerns. Frontend and backend code cannot accidentally import from each other, guaranteeing clean architectural boundaries at the package level.

Unified CI/CD Deployments

This monorepo structure guarantees unified CI/CD deployments. A single pipeline builds, tests, and deploys both namespaces in lockstep, eliminating version drift and ensuring atomic releases across the entire stack.