High-level architecture (components)

1. **Frontend (web & mobile)** — UI, local media capture, signaling client, WebRTC PeerConnections.
2. **Signaling server** — WebSocket/Socket.IO (for exchanging SDP/ICE, presence, invites).
3. **Media layer** — SFU (recommended) or MCU or pure P2P depending on scale; handles forwarding, recording hooks.
4. **STUN/TURN** — NAT traversal and relaying (production needs TURN).
5. **Recording/Storage** — server-side recording (SFU + recorder or ghost participant) → object store (S3).
6. **Backend APIs & DBs** — user, profiles, friend requests, social graph, feed, messages (persisted).
7. **Realtime data stores** — Redis for presence, rate limits, queues.
8. **Auth, Security, Privacy** — OAuth2 / OpenID Connect, JWT, TLS, optional E2EE for calls.
9. **Infra & scaling** — Kubernetes, autoscaling, monitoring, CDNs, load balancers.

Recommended tech stack (concrete)

Frontend

- React + Next.js (SSR for pages/profile SEO) + TypeScript. Tailwind CSS for rapid styling.
- Mobile: React Native or Expo (code reuse).
  Realtime / Signaling
- Node.js + TypeScript with Socket.IO or native WebSocket. Use NestJS or Express for REST + WebSocket.
  Media server (video conferencing)
- Pick an SFU: **mediasoup**, **Jitsi Videobridge**, or **Janus**. SFUs scale better than MCU and let you record centrally. (SFU vs MCU explained below). mediasoup.org+1
  NAT traversal
- **coturn** as TURN/STUN server (widely used open source). Production MUST have TURN to support restrictive NATs. Medium+1
  Recording
- Server-side recording by connecting a recorder to the SFU or using an SFU-built recording API (or a "ghost" participant that receives streams and writes to disk), then store files in S3 (or S3-compatible). Server-side recording requires a media server. BlogGeek.me+1
  Databases
- **Postgres** for relational data (users, auth, posts).
- **Neo4j** or a graph layer (or use Postgres with adjacency tables) for efficient mutual-connection/friend-recommendations. Use whichever team is comfortable with — Neo4j is optimized for graph queries. Medium+1
- **Redis** for presence, session store, rate-limiting, pub/sub.

- **Elasticsearch** for text search (profiles, posts) if you need search.
  Storage & CDN
- S3 (or Cloud object store) for recordings / uploaded media + Cloud CDN in front for global delivery.
  Auth & Notifications
- OAuth2 / OIDC for third-party login (Google/FB), JWT for API auth, refresh tokens. Push: FCM (Android) + APNs (iOS). Email: transactional service (SendGrid/Mailgun). Conviso AppSec
  Infra / Deployment
- Docker + Kubernetes for services, Helm charts, Traefik/NGINX LB, Cert-manager for TLS. Use managed DB (RDS) + managed object storage to reduce ops.
  Monitoring & Observability
- Prometheus + Grafana, Sentry for errors, ELK/Opensearch for logs.
  CI/CD
- GitHub Actions / GitLab CI → build, test, deploy.

Key architecture decisions & why

- **SFU vs MCU vs P2P**: SFU forwards individual participant streams (low CPU on server, scales well), MCU mixes streams server-side (higher CPU, simpler for clients), P2P works for small calls only. For scalable conferencing with recording, choose an SFU. DEV Community+1
- **Signaling**: Use WebSocket for low-latency signaling and presence. WebRTC itself needs signaling only to exchange SDP/ICE; media goes via peer or SFU. videosdk.live
- **TURN is mandatory in production**: Many corporate/phone networks block P2P; TURN (coturn) relays media for those clients. Medium+1
- **Server-side recording**: Implement via SFU hooks (record incoming RTP streams) or run a ghost participant that receives streams and writes them. Client-side recording is easier but less reliable/trustworthy. BlogGeek.me+1

Security & privacy (non-negotiable)

- Use HTTPS + secure WebSockets (wss). WebRTC uses DTLS/SRTP for media by default — keep libs patched. Consider E2EE if you need true end-to-end private calls (harder with server recording). Medium+1
- Protect recordings (encrypted at rest), implement RBAC for downloads, consent UI for recording. Comply with GDPR/CCPA as applicable.
- Rate-limit invites/requests, validate uploads, scan for malware.

MVP (practical phased roadmap)

1. **Phase 0 — auth + profiles + friend system**
   - User sign up/login, search, send/accept friend requests, basic profile pages. Use Postgres; implement friend requests as join table.
2. **Phase 1 — 1:1 chat + presence**

   - Real-time text with WebSocket, Redis presence, message storage. Notifications (push/email).

3. **Phase 2 — 1:1 video call (P2P) + STUN/TURN**
   - Implement WebRTC PeerConnection between two users. Add coturn. Record locally as optional feature.
4. **Phase 3 — group calls with SFU + server recording**
   - Deploy SFU (mediasoup/jitsi), add server-side recording, storage to S3, playback UI.
5. **Phase 4 — social features**
   - Mutual friend recommendations, feeds, follow vs friend semantics, search. Use graph DB if needed.
6. **Phase 5 — scale & polish**
   - Deploy on k8s, autoscaling, monitoring, analytics, E2EE options, content moderation pipelines.

Scalability & ops tips

- Start small: P2P for 1:1, then add SFU when you need >2 participants or server-side recording.
- Use managed services where ops cost > dev value (managed DB, object storage, managed TURN if needed).
- Measure bandwidth/CPU for SFU—video is expensive. Plan codecs and adaptive bitrate. mediasoup.org

Estimated minimal infra/services for an MVP

- Node.js app server (signaling + REST)
- coturn server (1–3 nodes)
- One SFU node (mediasoup/Jitsi) for early testing
- Postgres + Redis
- S3 (or MinIO for dev)
- K8s cluster or single VM for MVP

Helpful libraries & projects to explore

- mediasoup (SFU), Jitsi Meet/JVB, Janus (media servers). Clover Dynamics+1
- coturn (TURN server). Medium
- socket.io or ws for signaling.
- Simple Peer (browser WebRTC helpers), adapter.js for cross-browser WebRTC shims.

Quick checklist (dev → production)

- SSL/TLS everywhere
- TURN deployed + secured credentials
- Server-side recording + encrypted storage
- Auth: OAuth/OIDC + refresh token rotation
- Rate-limits, abuse detection, content moderation
- Monitoring + alerting (errors, bandwidth)
- Legal: privacy policy + recording consent