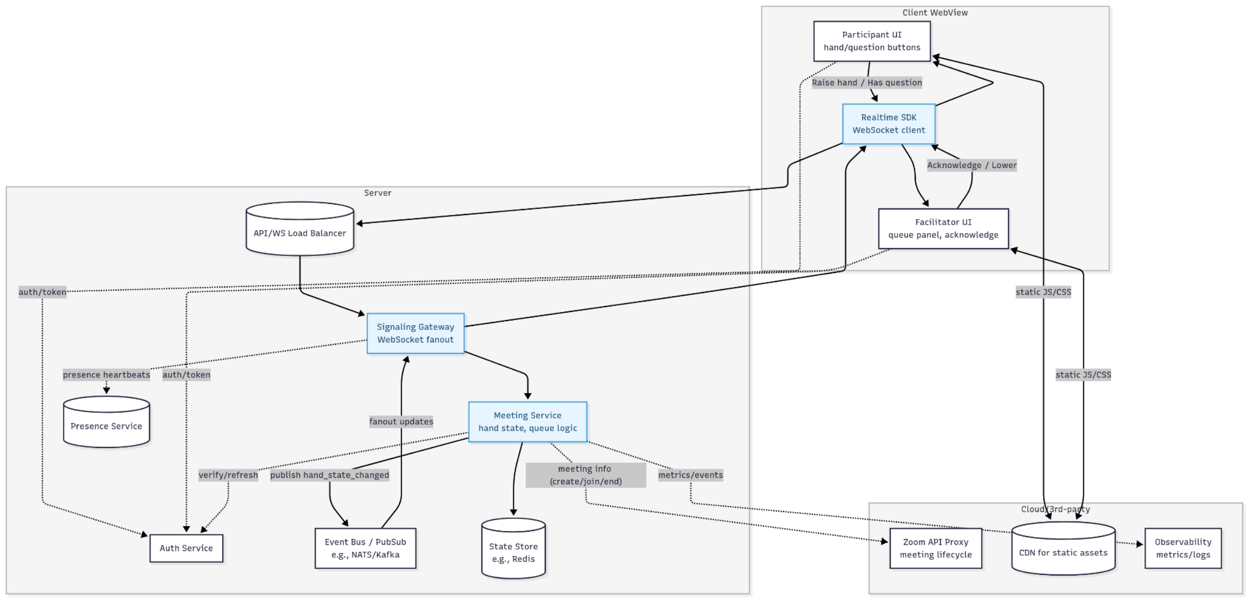
User Story 9: Question Queue Visibility

- Story: "As a user, I want a clear visual indicator when someone raises their hand or asks a question so that I can acknowledge all participants in a timely manner."
- What it's about: Helping meeting facilitators manage participation by making it obvious when attendees need attention, preventing people from being overlooked.

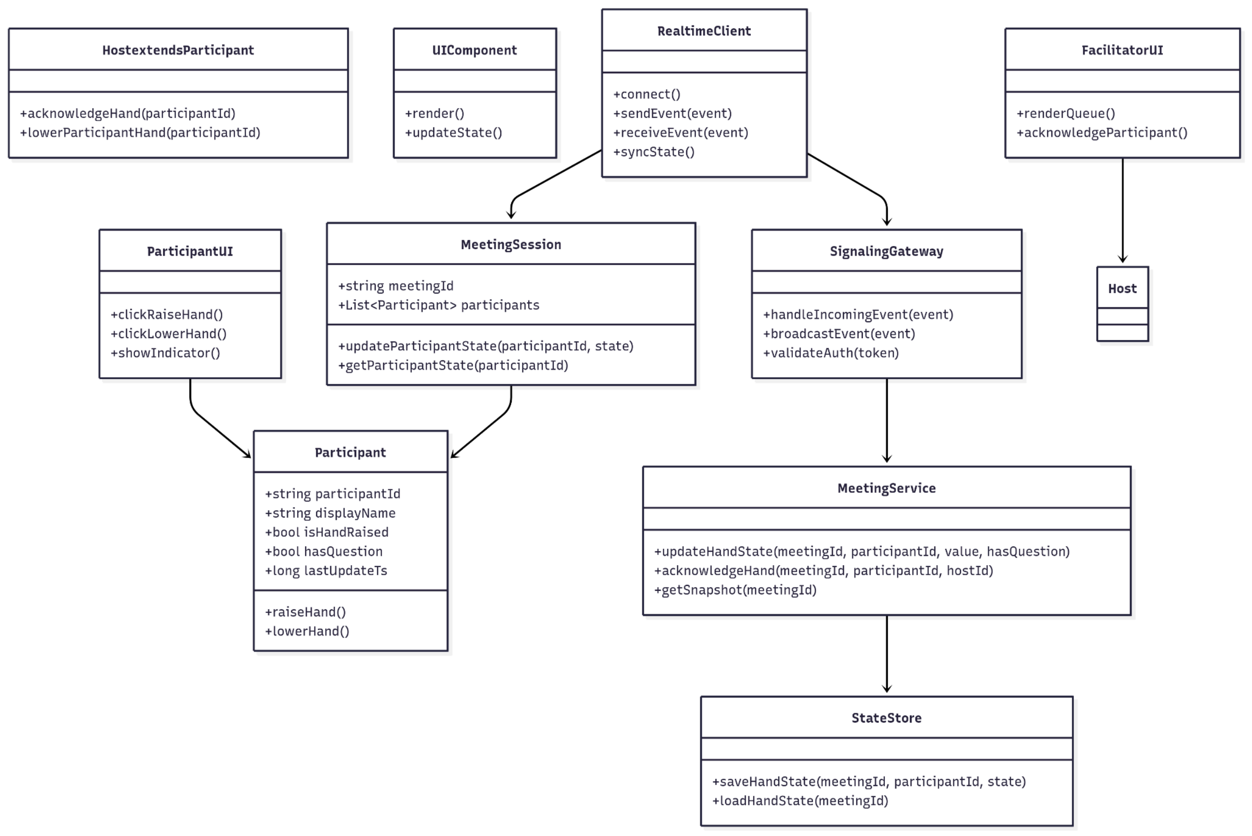
Header

Feature Name: Raised-Hand & "Has a Question" Indicator
Version: 1.0
Date: September 22, 2025
Author: Arthur Chien
Role: Software Engineer
Priority: High
Estimated Development Time: 2-3 sprints

Architecture Diagram



Class Diagram



List of Classes

Client-Side Classes

- **Participant**
  Represents an attendee in the meeting.
  - **Attributes: participantId, displayName, isHandRaised, hasQuestion, lastUpdateTs.**
  - **Methods: raiseHand(), lowerHand().**

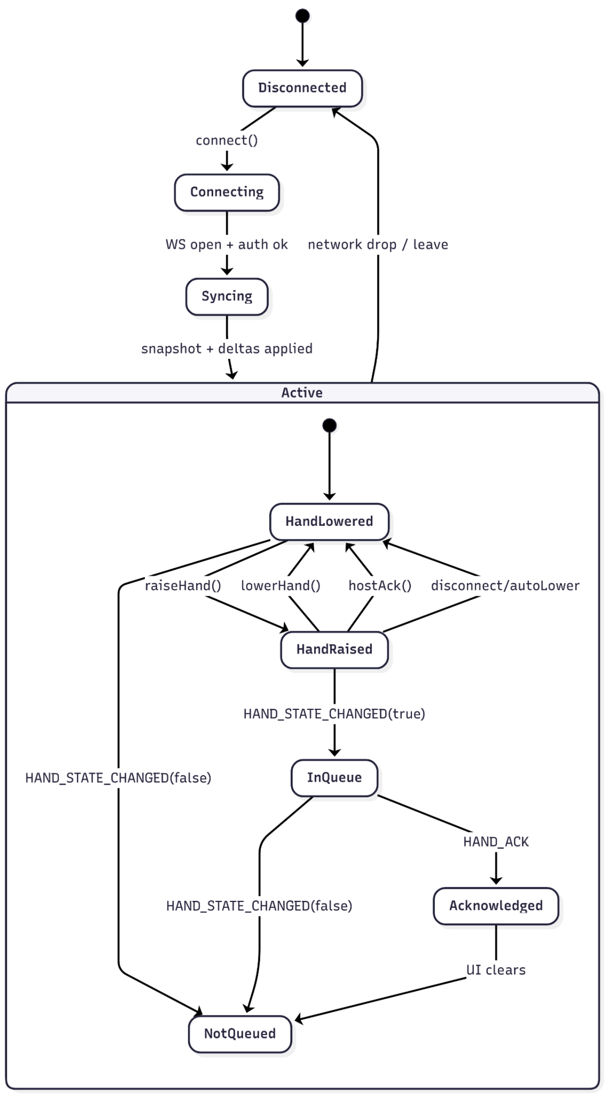
- **Host (extends Participant)**
  Special role with moderation privileges.
  - **Methods: acknowledgeHand(participantId), lowerParticipantHand(participantId).**
- **UIComponent (abstract)**
  Base class for visual components.
  - **Methods: render(), updateState().**
- **ParticipantUI (extends UIComponent)**
  UI elements for participants to raise/lower hand or ask question.
  - **Methods: clickRaiseHand(), clickLowerHand(), showIndicator().**
- **FacilitatorUI (extends UIComponent)**
  Host-facing UI for managing raised hands/questions.
  - **Methods: renderQueue(), acknowledgeParticipant().**
- **MeetingSession**
  Local representation of meeting state.
  - **Attributes: meetingId, participants.**
  - **Methods: updateParticipantState(), getParticipantState().**
- **RealtimeClient**
  Manages WebSocket connection and event sync.
  - **Methods: connect(), sendEvent(event), receiveEvent(event), syncState().**

Server-Side Classes

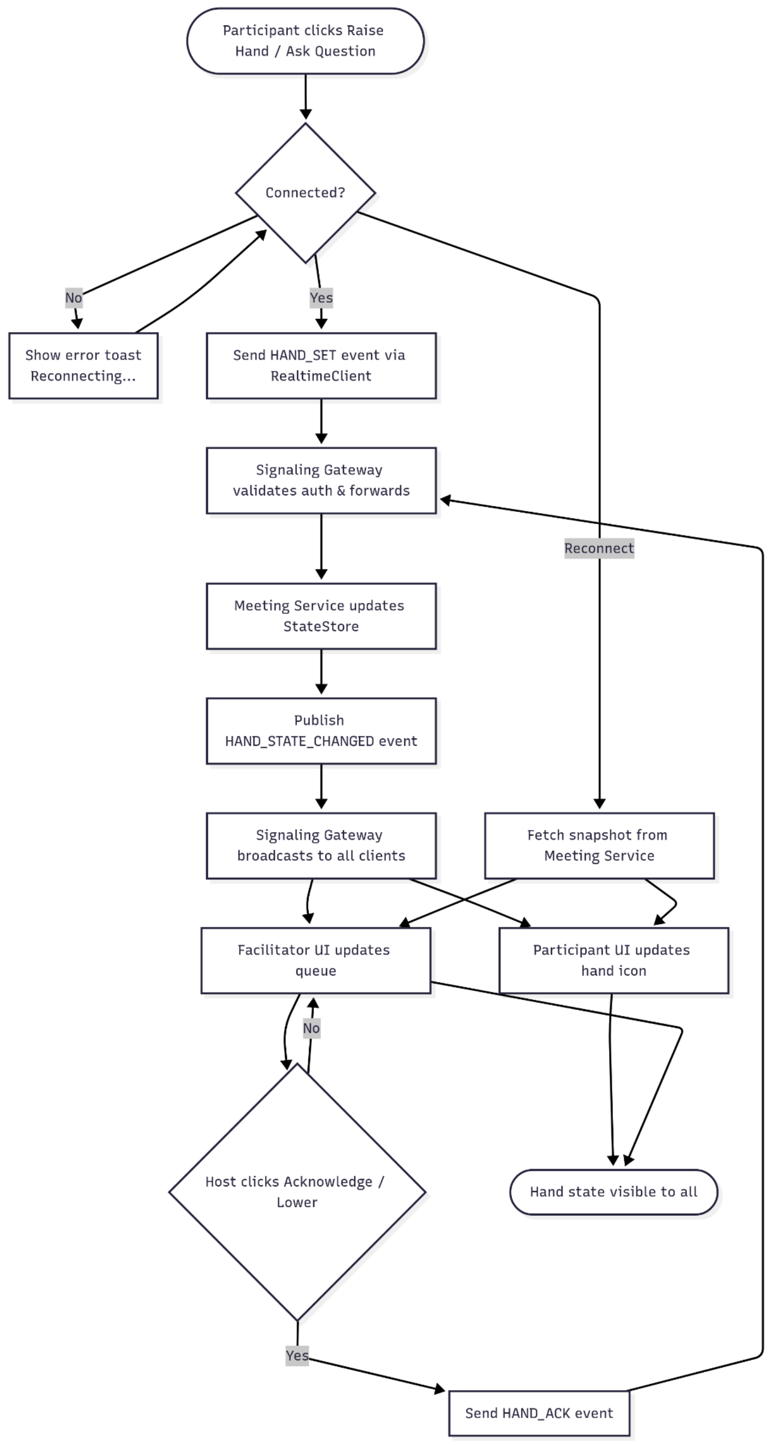
- **SignalingGateway**
  Entry point for client WS connections.
  - **Methods: handleIncomingEvent(event), broadcastEvent(event), validateAuth(token).**

- **MeetingService**
  Business logic for hand/question state changes.
  - **Methods: updateHandState(meetingId, participantId, value, hasQuestion), acknowledgeHand(meetingId, participantId, hostId), getSnapshot(meetingId).**
- **StateStore**
  Persists and retrieves hand/question states.
  - **Methods: saveHandState(meetingId, participantId, state), loadHandState(meetingId).**

State Diagrams



Flowcharts


Participant clicks Raise Hand / Ask Question
Connected?
No
Yes
Show error toast Reconnecting...
Send HAND_SET event via RealtimeClient
Signaling Gateway validates auth & forwards
Reconnect
Meeting Service updates StateStore
Publish HAND_STATE_CHANGED event
Signaling Gateway broadcasts to all clients
Fetch snapshot from Meeting Service
Facilitator UI updates queue
Participant UI updates hand icon
No
Host clicks Acknowledge / Lower
Hand state visible to all
Yes
Send HAND_ACK event

Development Risks and Failures Analysis

Client Layer Component Risks

ParticipantUI / FacilitatorUI

- UI Latency: Indicator updates may appear delayed if rendering is not optimized (e.g., re-rendering entire participant grid instead of diff updates).

- Accessibility Gaps: Missing ARIA live regions or keyboard shortcuts could exclude participants with disabilities.

- Inconsistent UX Across Devices: WebView differences (desktop vs. mobile) may cause mismatched behavior (e.g., tap gestures vs. click).

- Event Storming: Rapid toggling (raise/lower repeatedly) could overwhelm the UI or cause flickering indicators if debouncing is missing.

MeetingSession (local state)

- State Drift: If missed deltas are not reconciled properly during reconnect, a participant's hand state may be wrong until manually refreshed.

- Memory Leaks: Stale participant objects not pruned after disconnects could degrade performance in large meetings.

RealtimeClient

- Connection Drops: Flaky network can leave UI in inconsistent state if retries or snapshot syncs fail.

- Duplicate Events: If retry logic is not idempotent, user may see ghost indicators (e.g., two hands raised for the same participant).

- Cross-Version Compatibility: Older clients may not interpret new event fields (hasQuestion, ackedBy) correctly, leading to partial functionality.

Server Layer Component Risks

SignalingGateway

- Scaling Bottlenecks: If a single gateway node handles too many concurrent WS sessions, broadcast latency could exceed SLA.

- Auth Validation Lag: Extra auth calls on each event may cause raised-hand signals to delay beyond target latency.

- Fanout Failure: Partial broadcast failures could cause only a subset of clients to receive state updates.

MeetingService

- Ordering Guarantees: If sequence numbers are not strictly monotonic, clients may mis-order events and show wrong queue order.

- Business Logic Errors: Incorrect handling of HAND_ACK could leave participant hand stuck in raised state after acknowledgment.

- Snapshot Staleness: If StateStore is not updated atomically, reconnecting clients may receive outdated hand states.

StateStore

- Single Point of Failure: If Redis (or chosen store) fails, the service cannot persist or reload hand states.

- Write Contention: High-frequency raise/lower events could cause race conditions unless updates are atomic.

- TTL Misconfigurations: If states expire too early, raised hands may disappear prematurely.

Event Bus / PubSub

- At-Least-Once Semantics: Duplicate messages can occur without deduplication, leading to repeated hand-raise notifications.

- Dropped Messages: Network partitions between MeetingService and Gateway may silently drop events unless retry logic is robust.

Technology Stack

Client Layer

- **WebView Front-End**
  - **Framework**: React (for rendering participant grid, indicators, and queue panels).
  - **State Management**: Redux or Context API for hand/question state per participant.
  - **Accessibility**: ARIA live regions, keyboard bindings, high-contrast mode.
  - **Transport**: WebSockets via RealtimeClient for event delivery.

Server Layer

- **SignalingGateway**
  - **Runtime**: Node.js or Go for handling high-concurrency WebSocket connections.
  - **Load Balancer**: Nginx / Envoy for distributing WS sessions.
  - **Protocol**: Secure WebSockets (WSS) with OAuth2 token validation.
- **MeetingService**
  - **Runtime**: Java / Go / Node.js microservice for event processing.
  - **Framework**: gRPC or REST endpoints for control ops; WS events for real-time state.
  - **Logic**: Maintains queue ordering, enforces host permissions, manages acknowledgments.
- **StateStore**
  - **DB Choice**: Redis (in-memory, low-latency) for hand/question state keyed by meetingId.
  - **Schema**: hand_state:{meetingId} → {participantId, value, hasQuestion, ts}.

    - **Durability**: RDB/AOF persistence enabled to survive restarts.
- **Event Bus / PubSub**
    - **Option A**: NATS for lightweight, low-latency fanout.
    - **Option B**: Kafka for stronger persistence & replay (if at-scale needed).
    - **Responsibility**: Ensures ordered fanout of HAND_STATE_CHANGED events.

Cloud & Integration

- **CDN**: CloudFront / Akamai for delivering front-end assets.
- **Auth**: OAuth2 / OpenID Connect provider (used for validating WS tokens).
- **Zoom API Proxy**: Wraps Zoom-like APIs for meeting lifecycle; not extended here.
- **Monitoring**: Prometheus + Grafana for metrics (latency, error rates, event throughput).
- **Logging**: ELK stack (Elasticsearch, Logstash, Kibana) for auditing host actions (e.g., acknowledgments).

APIs

WebSocket Events (Realtime Channel)

Client → Server

1. **HAND_SET**

```json
{

 "type": "HAND_SET",

 "meetingId": "m_123",
 "participantId": "p_456",
 "value": true,                 // true = raised, false = lowered
 "hasQuestion": true,           // optional, indicates question intent
 "ts": 1737783000123,
 "idempotencyKey": "hash123"
}
```

2. **HAND_ACK** (Host only)

```JSON
{
  "type": "HAND_ACK",
  "meetingId": "m_123",
  "targetParticipantId": "p_456",
  "hostId": "h_789",
  "ts": 1737783000456,
  "idempotencyKey": "hash124"
}
```

Server → Client

1. **HAND_STATE_CHANGED**

```JSON
{
  "type": "HAND_STATE_CHANGED",
  "meetingId": "m_123",

  "participantId": "p_456",

  "value": true,                // current state

  "hasQuestion": false,

  "ackedBy": null,              // or hostId if acknowledged

  "reason": null,               // "offline" | "manual"

  "seq": 1042,

  "ts": 1737783000150

}
```

2. **SNAPSHOT** (on join/reconnect)

```
JSON
{

  "type": "SNAPSHOT",

  "meetingId": "m_123",

  "seqHead": 1042,

  "handState": [

    {"participantId":"p_1","value":true,"hasQuestion":false,"ts":1737782999000},

    {"participantId":"p_2","value":false,"hasQuestion":false,"ts":1737782998000}

  ]

}
```

REST / gRPC APIs (Server Control Plane)

1. **GET /meetings/{meetingId}/handState**

- Returns full snapshot of current hand/question states.
- Used for debug/admin or fallback if WS fails.

2. **POST /meetings/{meetingId}/participants/{participantId}/hand**

- Body: { "value": true, "hasQuestion": false }
- Updates participant's hand/question state (non-realtime path).

3. **POST /meetings/{meetingId}/participants/{participantId}/ack**

- Body: { "hostId": "h_789" }
- Host acknowledgment; clears participant's raised hand.

Auth & Security

- **All WS events** must include OAuth2 bearer token on connect; validated against meetingId.
- **Host-only actions** (HAND_ACK) require role claim role: host|cohost.
- **Rate Limits**:
    - Max 5 raise/lower events per 10 seconds per participant.
    - Host ACK limited to 10 per second.

Public Interfaces

Client-Facing Interfaces

1. Participant UI Controls

- **Button**: *Raise Hand / Lower Hand*

    - Toggle control bound to raiseHand() / lowerHand().
    - Visible state: icon highlighted when active.
- **Button**: *Ask Question* (optional variant of raise hand)
    - Sets hasQuestion = true along with raised state.
- **Indicator**:
    - Hand icon or badge shown on participant's tile.
    - Tooltip: "Raised hand" or "Has a question."
- **Accessibility**:
    - Keyboard shortcut: Alt+Y (toggle).
    - ARIA live region: " raised hand."

2. Facilitator (Host) UI Controls

- **Queue Panel**:
    - Ordered list of participants with raised hands.
    - Metadata: participant name, time raised, question flag.
- **Actions**:
    - *Acknowledge* → clears raised hand, optional toast shown.
    - *Lower Hand* → force clears state.
- **Indicator**:
    - Count badge (e.g., "3 hands raised").

Server-Facing Interfaces

1. WebSocket Endpoints

- wss://<gateway>/realtime?token=<auth>
  - Input: HAND_SET, HAND_ACK messages.
  - Output: HAND_STATE_CHANGED, SNAPSHOT messages.

2. REST Endpoints (optional fallback / admin)

- GET /meetings/{meetingId}/handState → JSON snapshot.
- POST /meetings/{meetingId}/participants/{id}/hand → set state.
- POST /meetings/{meetingId}/participants/{id}/ack → host ack.

External Interfaces

- **Zoom API Proxy**
  - Provides meeting lifecycle (join/leave, auth validation).
  - Not extended for hand-raise logic; only ensures participant/host roles.
- **Monitoring / Logging Dashboards**
  - Prometheus metrics exposed:
    - hand_event_latency_ms (p95, p99)
    - hand_event_error_rate
    - active_hands_count per meeting
  - Logs include: participant actions, host acknowledgments, error traces.

Data Schemas

1. In-Memory / StateStore Schema

Key: hand_state:{meetingId}
Value: Hash/Map of participant hand states

```json
{
 "participantId": "p_123",
 "isHandRaised": true,
 "hasQuestion": false,
 "ackedBy": null,
 "lastUpdateTs": 1737783000123
}
```

- **participantId**: Unique ID of the participant in meeting.
- **isHandRaised**: Boolean flag for raised hand.
- **hasQuestion**: Boolean flag if participant indicates a question.
- **ackedBy**: Host/co-host ID if acknowledgment cleared the hand.
- **lastUpdateTs**: Server-side timestamp of last state change.

---

2. Event Schema (PubSub / WS Broadcasts)

HAND_STATE_CHANGED Event

```json
{
 "type": "HAND_STATE_CHANGED",
 "meetingId": "m_123",
 "participantId": "p_123",
 "isHandRaised": true,
 "hasQuestion": false,
 "ackedBy": null,
 "reason": null,
 "seq": 1042,
 "ts": 1737783000150
}
```

SNAPSHOT Event

```JSON
{
  "type": "SNAPSHOT",
  "meetingId": "m_123",
  "seqHead": 1042,
  "handState": [
    {
      "participantId": "p_123",
      "isHandRaised": true,
      "hasQuestion": false,
      "ackedBy": null,
      "lastUpdateTs": 1737782999000
    },
    {
      "participantId": "p_456",
      "isHandRaised": false,
      "hasQuestion": false,
      "ackedBy": null,
      "lastUpdateTs": 1737782998000
    }
  ]
}
```

---

3. Audit Log Schema (for compliance / debugging)

Stored in persistent DB or log pipeline (not user-facing).

```JSON
{
  "logId": "uuid-123",
  "meetingId": "m_123",
  "actorId": "h_789",
  "targetParticipantId": "p_123",
  "action": "HAND_ACK",
  "result": "success",
  "ts": 1737783000456
}
```

---

4. API Response Schema (REST Fallback)

GET /meetings/{meetingId}/handState

```JSON
{
 "meetingId": "m_123",
 "seqHead": 1042,
 "participants": [
   {
    "participantId": "p_123",
    "isHandRaised": true,
    "hasQuestion": false,
    "ackedBy": null,
    "lastUpdateTs": 1737782999000
   },
   {
    "participantId": "p_456",
    "isHandRaised": false,
    "hasQuestion": false,
    "ackedBy": null,
    "lastUpdateTs": 1737782998000
   }
 ]
}
```

Security and Privacy

Authentication & Authorization

- **OAuth2 Access Tokens**:
  - All WebSocket connections require a valid bearer token.
  - Token must include meetingId, participantId, and role.

- **Role Enforcement**:
    - Regular participants can only raise/lower their own hand.
    - Hosts/co-hosts can acknowledge or lower other participants' hands.

Data Integrity

- **Idempotency Keys**: Prevent duplicate event processing during retries.
- **Sequence Numbers**: Ensure consistent ordering of `HAND_STATE_CHANGED` events across all clients.
- **Replay Protection**: Events older than the last known `seq` are discarded by clients.

Transport Security

- **Encrypted Channels**: All signaling runs over WSS (WebSockets over TLS 1.2+).
- **HSTS / CSP**: Front-end enforces HTTPS with strong headers to prevent downgrade attacks.
- **Certificate Management**: Automated rotation (e.g., via Let's Encrypt or internal PKI).

Privacy Considerations

- **Minimal PII**: Events carry only `participantId` and display name (from existing meeting roster).
- **Ephemeral States**: Hand/question indicators are transient, tied to active meeting only.
- **Audit Logs**: Host acknowledgments are logged with `actorId` and `targetParticipantId` for accountability.

- **Data Retention**: Hand state logs expire after 30 days (configurable), unless retained for compliance.

Threat Mitigation

- **Flood Control**: Per-participant rate limits (e.g., max 5 hand raises per 10 seconds).
- **Spoofing Prevention**: Tokens signed by Auth service; cannot forge hand events for other participants.
- **Replay Attacks**: Each event timestamp (`ts`) and sequence number (`seq`) validated; stale or duplicate events rejected.
- **Denial of Service**: Gateway enforces connection caps and per-IP throttling.

Risks to Completion

Technical Risks

- **Realtime Reliability**: Achieving sub-500 ms end-to-end latency under load may require careful tuning of WebSocket fanout and Redis writes.
- **Ordering Guarantees**: Ensuring strict monotonic sequence numbers across distributed meeting service instances could introduce complexity.
- **Reconnect Sync**: Edge cases in snapshot + delta recovery may leave clients with stale or inconsistent hand states.
- **Cross-Client Compatibility**: Older or partially updated clients may not recognize `hasQuestion` or `ackedBy` fields, causing divergent UI behavior.

Integration Risks

- **Dependency on Auth Service**: Delays or failures in OAuth2 token validation can block event flow.
- **Interaction with Zoom API Proxy**: If participant roles are not synced correctly, a user may gain or lose acknowledgment privileges incorrectly.
- **Event Bus Backpressure**: If PubSub queues lag, broadcasts may be delayed or dropped.

Delivery Risks

- **Limited Testing at Scale**: Without large test meetings (100–300 participants), concurrency bugs may only surface in production.
- **Mobile WebView Variability**: Differences in how Android/iOS handle WebSocket reconnections could cause inconsistent experience.
- **Accessibility Gaps**: If ARIA live regions or shortcuts are not fully tested, feature may fail accessibility audits.

Resource & Timeline Risks

- **Redis Persistence Tuning**: Enabling durability may affect latency; requires dedicated ops time.
- **Operational Overhead**: Additional monitoring and alerting pipelines may not be ready by release.
- **Team Familiarity**: If the team lacks prior experience with high-concurrency WS systems, learning curve may extend delivery.

LLM Chat Log: https://chatgpt.com/share/68d47b28-bc90-800b-84ec-279a63113a33