LAN Collaboration Suite Technical Reference

This document rebuilds the project documentation from the ground up. It enumerates every functional requirement, maps each capability to the concrete Python and JavaScript symbols that implement it, and highlights operational details down to the "teeny tiny" UX flourishes—such as inline @ mentions, duplicate guards, and status badges.

1. Requirements & Feature Traceability

Requirement	Implementation Highlights
Real-time video conferencing	`server/video_server.py` (UDP relay), `client/video_client.py` (capture/receive), `assets/main.js` (`ensureVideoTile`, `renderVideoFrame`).
Real-time audio conferencing	`server/audio_server.py` (mix loop), `client/audio_client.py` (capture/playback), presence badges in `assets/main.js` (`updatePeerMedia`).
Screen sharing with presenter control	`server/screen_server.py`, `client/screen_client.py::ScreenPublisher`, UI toggles in `assets/main.js` (`togglePresent`, `updateStagePresenterState`).
Group chat with Discord-style mentions	Control plane via `server/control_server.py`, chat persistence in `server/session_manager.py::add_chat_message`, inline mention system in `assets/main.js` (`handleChatInputChange`, `renderChatInputOverlay`, `insertMention`, `parseMentions`).
File upload/download	`server/file_server.py`, `client/file_client.py`, drag-and-drop UX in `assets/main.js` (`handleDrop`, `uploadFilesSequentially`).
Presence, reactions, hand raise, typing indicators	`session_manager.py` presence payloads, `control_server.py::_handle_message`, `assets/main.js` (`updatePresenceEntry`, `handleReactionSelection`).
Latency monitoring	`server/latency_server.py`, `client/app.py::LatencyProbe`, latency badge styling in `assets/styles.css`.
Administrative controls	REST API in `server/admin_dashboard.py`, front-end assets under `adminui/`, shutdown orchestration in `session_manager.py::disconnect_all`.

Requirement	Implementation Highlights
Deployment deliverables	PyInstaller specs under `build/spec/`, automation script `scripts/build_executables.py`.

2. Runtime Topology & Transports

```
                                +--------------------------+
                                |        Browser UI        |
                                +-------------+------------+
                                                            |
                                                            | HTTP / WebSocket (loopback)
                                                            v
    +---------------------------+---------------------------+
    |                   Client Runtime                      |
    |  - FastAPI app (`client/app.py`)                      |
    |  - Media capture (audio/video/screen)                 |
    |  - File transfer helper                               |
    +---------------------------+---------------------------+
                                                            |
                                                            | LAN sockets (TCP/UDP)
                                                            v
                                     +----------+-----------+
                                     |     Collaboration    |
                                     |        Server        |
                                     +----------------------+
```

Channel	Transport	Source -> Destination	Key Symbols
Control & chat	TCP	`client/control_client.py` <-> `server/control_server.py`	`ControlAction` enum, `SessionManager.broadcast()`
Audio	UDP	`AudioClient` <-> `AudioServer`	`MediaFrameHeader`, `_mix_loop()`
Video	UDP	`VideoClient` <-> `VideoServer`	`stream_id_for()`, `VideoServer.datagram_received()`
Screen sharing	TCP	`ScreenPublisher` -> `ScreenServer` -> viewers	`_read_frame()`, `ControlAction.SCREEN_FRAME`
Files	TCP	`FileClient` <-> `FileServer`	`_handle_upload()`, `_handle_download()`
Latency probes	UDP	`LatencyProbe` <-> `LatencyServer`	`_LatencyProtocol`, `LatencyProbe._send_probe()`
UI bridge	Loopback WebSocket	Browser <-> `ClientApp.WebSocketHub`	`WebSocketHub.broadcast()`

Reliability design: - TCP channels guarantee ordering for chat, files, and screen control. Backpressure is absorbed by asyncio streams. - UDP media deliberately trades reliability for latency. Each client has local jitter buffers (`AudioClient._play_queue`, `VideoClient._peers`). - The UI stays responsive during control drops because static assets come from the local FastAPI server.

3. Control & Media Flows

3.1 Join Sequence

1. Browser loads assets from the embedded client server (`ClientApp._configure_routes`).
2. `ControlClient.connect()` opens TCP, sends `HELLO`, and waits for `ControlAction.WELCOME`.
3. `ControlServer._handle_client()` registers the username via `SessionManager.register()` and replies with chat history, presence, files, media config, and time limit state.
4. `ClientApp` starts media helpers (`VideoClient`, `AudioClient`, `ScreenPublisher`) on demand and schedules heartbeats/latency probes.

3.2 Media Lifecycle

- **Audio**: `AudioClient.set_capture_enabled()` toggles microphone capture. Frames reach `AudioServer.datagram_received()`, are queued per user, and `_mix_loop()` sends mixed streams back.
- **Video**: `VideoClient._capture_loop()` encodes JPEG frames, sends them, and replays them locally so the UI shows instant feedback. Peers are tracked with `VideoClient.update_peers()`.
- **Screen sharing**: `ScreenPublisher.start()` opens a TCP stream, sends metadata, and pushes JPEG frames. `ScreenServer` broadcasts `SCREEN_CONTROL` and `SCREEN_FRAME` events through the control plane.

3.3 Chat & Mentions

- `assets/main.js` maintains chat state and callouts. Core mention helpers:
  - `handleChatInputChange()` detects `@` sequences, filters matches, and renders the autocomplete list.
  - `renderChatInputOverlay()` mirrors the raw input with inline `<span class="mention-token">` chips.
  - `insertMention()` injects the mention text, prevents duplicates via `parseMentions()`, and triggers `flashMentionToken()` animation.
  - `parseMentions()` accepts alphanumeric, underscore, and hyphenated usernames and cross-checks `participants` to avoid stray tokens.
- Submitted messages go through `ControlClient.send_chat()`, with targeted delivery handled by `ControlServer._handle_message()`.

3.4 Files

- Drag-and-drop in `assets/main.js` (`handleDragEnter`, `handleDrop`) funnels into `FileClient.upload()`.
- `FileServer._handle_upload()` writes streaming chunks using `aiofiles`, updates progress via `ControlAction.FILE_PROGRESS`, and advertises completed offers (`FILE_OFFER`).
- Downloads stream chunk-by-chunk (`FileClient.download()` yields an `AsyncIterator`).

3.5 Administrative Controls

- `AdminDashboard` mounts the static admin UI and exposes JSON endpoints (`/api/state`, `/api/actions/kick`, `/api/actions/time-limit`).
- Snapshot responses delegate to `SessionManager.snapshot()`, which collates presence, chat history, events, latency stats, and shutdown status.
- Time limits rely on `SessionManager.set_time_limit()` and `get_time_limit_status()`. Client enforcement lives in `ClientApp._maybe_schedule_time_limit_leave()`.

4. Module & Function Inventory

4.1 Shared Layer (`shared/`)

Symbol	Responsibility	Notes
`ControlAction`	Enum of TCP control opcodes.	Used everywhere a control message is dispatched.
`ChatMessage`	Dataclass with `from_dict()`/`to_dict()` helpers.	Used by `SessionManager.add_chat_message()` and tests.
`ClientIdentity`	Represents HELLO payload.	Carries optional pre-shared key for secured deployments.
`MediaFrameHeader` / `MEDIA_HEADER_STRUCT`	Packed header for UDP frames.	Shared by audio and video paths.
`FileOffer`	Dataclass for sharing file metadata.	Broadcast after successful uploads.
`resource_paths.resource_path()`	Locates packaged assets (admin UI, storage).	Keeps PyInstaller builds working.

4.2 Session Management (`server/session_manager.py`)

Category	Key Functions	Behaviour
Registration & lifecycle	`register()`, `unregister()`, `disconnect_all()`, `ban_user()`, `unban_user()`, `list_clients()`	Manage the authorative client registry, enforce bans, and broadcast join/leave events.
Presence tracking	`_client_presence_payload()`, `get_presence_entry()`, `get_presence_snapshot()`, `snapshot()`	Produce data consumed by UI badges, admin dashboard, and mention autocomplete (`participants` list).
Media state	`update_media_state()`, `get_media_state_snapshot()`	Synchronize audio/video toggles across participants.
Chat & reactions	`add_chat_message()`, `get_chat_history()`, `get_chat_history_for()`	Persist the last 200 messages and filter by recipient for targeted chats.
Typing, hands, latency	`set_typing()`, `set_hand_status()`, `update_latency()`	Feed presence overlays and highlight cues (typing banner, raise-hand icon, latency badge).
Heartbeats	`mark_heartbeat()`, `heartbeat_watcher()`	Detect stalled control connections, prune stale entries, and broadcast forced leaves.
Time limits	`set_time_limit()`, `get_time_limit_status()`, `_build_time_limit_status_locked()`	Drive meeting countdowns and forced eject when the limit expires.
Admin utilities	`mark_shutdown_requested()`, `record_admin_notice()`, `get_recent_events()`, `record_blocked_attempt()`	Provide observability and enforce admin-initiated actions.

Helper: `_calculate_rate()` turns byte counters into bits-per-second for admin snapshots.

4.3 Control Plane (`server/control_server.py`)

Function	Purpose
`start()/stop()`	Bind/unbind the TCP server.
`_handle_client()`	Perform HELLO handshake, send `WELCOME`, process stream with `decode_control_stream()`, and cleanly unregister on exit.
`_handle_message()`	Dispatch every `ControlAction` (chat, media toggles, presenter grants, typing, reactions, latency updates).
`_broadcast_presence_entry()`	Refresh overlay data after local state changes.
`force_disconnect()`	Kick users on admin request, broadcast `USER_LEFT`, purge media sockets, and ban the username.

4.4 Media Servers

Audio (`server/audio_server.py`)

Function	Summary
`start()/stop()`	Manage UDP transport and the background `_mix_loop()` task.
`datagram_received()`	Handle registration handshakes and ingest PCM frames tagged with `MediaFrameHeader`.
`_enqueue()`	Buffer per-user frames while holding `_lock` to avoid race conditions.
`_mix_loop()`	Every 20 ms, mix all other users' frames and send individualized streams back.
`remove_user()`	Drop buffers and address bindings when a participant leaves.

Video (`server/video_server.py`)

Function	Summary
`start()/stop()`	Manage UDP socket lifecycle.
`datagram_received()`	Register senders and blindly relay encoded JPEG frames to every other peer.

Function	Summary
`remove_user()`	Purge address when control plane reports a disconnect.

Screen (`server/screen_server.py`)

Function	Summary
`start()/stop()`	Manage TCP listener.
`_handle_connection()`	Authenticate presenter (`SessionManager.is_presenter()`), broadcast `SCREEN_CONTROL` state, relay base64 frames as `SCREEN_FRAME`.
`_read_json()/ _read_frame()`	Framing helpers wrapping TCP byte streams in length-prefixed payloads.

Latency (`server/latency_server.py`)

Function	Summary
`LatencyServer.start()/stop()`	Spin up UDP responder with optional pre-shared key.
`_LatencyProtocol.datagram_received()`	Validate key, echo timestamps, and guard against malformed payloads.

4.5 File Transfer (`server/file_server.py`)

Function	Summary
`start()/stop()`	Bind TCP file port; cleanup storage on shutdown.
`_handle_upload()`	Receive metadata, stream chunks to disk, broadcast progress, and register `StoredFile`.
`_handle_download()`	Serve metadata then stream file contents; sends empty chunk to terminate.
`list_files()` / `get_file()`	Provide metadata for `FILE_OFFER` responses and downloads.
`cleanup_storage()`	Remove tracked files and stray artifacts; invoked on shutdown and by admin tasks.

4.6 Admin & Observability (`server/admin_dashboard.py`)

Symbol	Summary
`_InMemoryLogHandler`	Captures recent log lines for `/api/state` log tail.
`AdminDashboard`	FastAPI app exposing HTML front-end and JSON APIs (state snapshots, time limit, notices, kicks, shutdown, event export).
`AdminServer`	Background runner that boots Uvicorn, keeps it alive, and supports graceful stop.

4.7 Client Runtime (`client/app.py`)

Function/Class	Summary
`ClientApp.__init__()`	Initializes service clients, presence caches, WebSocket hub, reconnection state, and UI routes.
`_configure_routes()`	Mounts static assets, exposes REST endpoints (`/api/config`, `/api/files/*`), and sets up WebSocket control bridge.
`WebSocketHub`	Manages browser connections; `broadcast()` sends UI updates (chat, files, presence).
`LatencyProbe` (`start()`, `_send_probe()`, `_handle_packet()`)	Periodically measures round-trip delay and reports to control channel.
File APIs (`upload_file`, `download_file`)	Wrap `FileClient` calls, relay progress, enforce size limits (`MAX_UPLOAD_SIZE_BYTES`).
UI message handlers (`_handle_ui_message`, `_apply_server_event`, `_build_snapshot`)	Synchronize client state with WebSocket UI.
Reconnect logic (`_schedule_reconnect`, `_connect_control`)	Implements exponential backoff and status banners.
Time limit enforcement (`_apply_time_limit`, `_maybe_schedule_time_limit_leave`)	Aligns client shutdown with server-side timers.

4.8 Client Control Plane (`client/control_client.py`)

Function	Summary
`connect()`	Open TCP stream, send `HELLO`, spin up send/receive coroutines, start heartbeat loop.
`send()` / `send_chat()` / `send_typing()` / `send_hand_status()` / `send_reaction()` / `send_latency_update()`	Encode `ControlAction` messages and enqueue them for `_send_loop()`.
`_recv_loop()`	Decode control stream, dispatch to callbacks, manage disconnect reasons.
`_heartbeat_loop()`	Send periodic `HEARTBEAT` actions.

4.9 Client Media Helpers

- **`client/audio_client.py`**
  - `start()/stop()` configure audio streams and UDP transport.
  - `_capture_callback()` and `_playback_callback()` interface with `sounddevice` to ship/consume PCM frames.
  - `set_capture_enabled()` toggles mic capture without tearing down playback.
- **`client/video_client.py`**
  - `start()/stop()` manage UDP endpoint and capture task.
  - `_capture_loop()` reads OpenCV frames, encodes JPEG, emits base64 for UI rendering, and sends binary frames to server.
  - `update_peers()` maps CRC32 `stream_id` values to usernames for downstream decoding.
- **`client/screen_client.py`**
  - `start()/stop()` wrap asynchronous screen capture loop.
  - `_run()` handles handshake, capture via `mss`, JPEG encoding, and send cadence.
  - `_write_frame()` and `_send_json()` enforce length-prefixed framing.
- **`client/file_client.py`**
  - `upload()` streams multipart data, invokes optional progress callback, and handles server acknowledgements.
  - `download()` returns metadata plus an async generator for chunked consumption.

4.10 Front-End Assets (`assets/`)

Symbol	Summary
`renderChatInputOverlay()`	Mirrors chat text into overlay with mention chips and placeholder fallback.
`handleChatInputChange()`	Drives mention autocomplete, filters out false positives (emails), and syncs overlay.
`insertMention()`	Inserts deduplicated mentions, focuses cursor, and triggers highlight animation.
`parseMentions()`	Extracts valid usernames using `/@([A-Za-z0-9_-]+)/g` and cross-references the participant set.
`flashMentionToken()`	Adds the `.flash` CSS class so repeated mentions pulse instead of duplicating.
`setupNotificationBadges()` / `clearNotificationsForActiveTab()`	Manage chat/file unread counters.
`updatePresenceEntry()`	Consolidates typing, hand-raise, and media status updates into a single UI refresh.
`handleReactionSelection()`	Sends emoji through control channel and replays them locally with rate limiting.

CSS (`assets/styles.css`) introduces: - Wider chat sidebar (max-width 380px) for mention readability. - `.mention-token` styling, overlay line-height synchronization, and `.flash` animation (`@keyframes mentionFlash`). - Transparent input background to let overlay text replace the raw textarea text.

4.11 Tests (`tests/`)

File	Coverage
`test_protocol.py`	Validates control message encoding/decoding and dataclass helpers.
`test_session_manager.py`	Exercises registration, chat history filtering, time limits, hand raises, and latency updates.
`test_control_server.py`	Integration-style tests ensuring HELLO handshakes, message routing, and bans behave.
`test_client_time_limit.py`	Ensures client countdown logic respects expiring limits.

5. UX Details & Micro Features

- Mention popup sticks to caret location, supports hyphenated names, and avoids ghost overlays by toggling `hidden` class and syncing overlay text.
- Duplicate mentions flash in place instead of re-inserting (`flashMentionToken`).

- Typing indicators collapse multiple users into friendly strings ("Alice and Bob are typing…").
- Latency badge color-codes thresholds (<=120 ms good, <=250 ms warn, otherwise red) and shows jitter when available.
- Drag-and-drop overlay (`#drag-overlay`) prevents accidental uploads and guides users before joining.
- Chat/file tab badges avoid spamming the user once the pane is active (`suppressChatNotifications`).

6. Build & Deployment Notes

- `scripts/build_executables.py` wraps PyInstaller invocations for both client and server specs under `build/spec/`.
- `scripts/cluster_launcher.py` demonstrates multi-instance orchestration for lab testing.
- `pyproject.toml` pins runtime dependencies (FastAPI, sounddevice, OpenCV, MSS, etc.) and includes optional extras for packaging.

7. Operational Guidance

- Start server: `python -m server --host 0.0.0.0 --port 8765 --admin-port 8080` (match the actual CLI options defined in `server/__main__.py`).
- Start client: `python -m client --server 192.168.1.10 --username Alice`. Browser UI launches automatically (`webbrowser.open()` in `ClientApp.start_local_ui()`).
- Use firewall rules to scope UDP ports for audio/video/latency if the LAN is shared.
- Shutdown best practice: call `/api/actions/shutdown` from the admin UI; this triggers `SessionManager.disconnect_all()` so no socket leaks remain.

8. Testing & Validation Workflow

- Unit tests can be executed with `pytest`. Focus on control/session tests after protocol changes and run `test_client_time_limit.py` when adjusting countdown logic.
- Manual smoke checks:
  1. Join two clients, toggle mic/video, verify presence updates.
  2. Type `@` to confirm mention autocomplete, ensure duplicate prevention by clicking the same user twice.
  3. Upload a file and verify broadcast to remote peer plus admin dashboard storage stats.
  4. Trigger admin time limit and confirm auto-disconnect message.

This reference should give new contributors and operators enough depth to understand not only where each feature lives, but also how its functions collaborate across the Python back-end and browser front-end.