WAITROOM — Technical Architecture Specification

Date: 2026-04-12

Prepared by: Waitroom

Table of Contents

The Coordination Layer for OpenClaw Agents

Date	April 12, 2026
Status	Internal
Audience	Engineering Team

Table of Contents

1. Problem Statement

OpenClaw is becoming the standard way people run AI agents — across WhatsApp, Slack, terminals, and dozens of other channels. But as agents do more consequential work, the ecosystem lacks a structured coordination layer between agents and humans. Today, agents either operate fully autonomously (creating risk) or are bottlenecked behind rigid workflow tools not built for agent volume, velocity, and context requirements.

Enterprises cannot confidently deploy agents at scale without governance infrastructure. Developers build custom approval logic from scratch for every agent. There is no shared protocol for the most fundamental interaction pattern: an agent needing a human to approve, reject, or modify a proposed action.

Waitroom is the coordination layer for OpenClaw agents. It organises all coordination into **rooms** — workspaces where agents and humans collaborate under clear rules. Each room has its own policies, agents, and trust scores. Waitroom provides: **identity** (which human owns this agent?), **capability grants** (what can it do autonomously, and what requires human sign-off?), **interaction rules** (can agent A delegate to agent B?), and **audit** (what did it actually do?). It is the ***permissions model for the agent era***, analogous to how mobile operating systems introduced app permissions for camera, location, and contacts.

Waitroom does not run agents. OpenClaw runs agents. Waitroom sits between OpenClaw and the humans who are ultimately responsible for what those agents do.

---

2. Goals

#	Goal	Success Metric	Status
G1	Sub-5-minute time to first approved check-in from signup	Median onboarding time < 5 minutes	✅ Built
G2	Single API serves MCP, REST, SDK, and CLI clients	100% API parity across all surfaces	✅ Built
G3	Real-time decision delivery: agents receive approvals/rejections within seconds of human action	P95 latency < 500ms (WebSocket), < 2s (SSE)	✅ Built
G4			

#	Goal	Success Metric	Status
	Immutable, compliance-ready audit trail for every coordination event	Every decision logged with timestamp, actor, and full context	✅ Built
G5	Dynamic trust scoring that reduces human approval burden over time	Agents with >90% approval rate auto-graduate to reduced check-in requirements	⚠️ Partial
G6	Multi-agent specialization: agents declare capabilities and receive best-match task routing	Tasks routed to best-fit agent with >80% accuracy	⚠️ Partial
G7	Agent-to-agent delegation: agents can post sub-tasks linked to parent tasks	Child task results flow back to parent agent via `parentCheckInId`	⚠️ Partial

> *G1–G4 are fully operational. G5: trust scores are tracked per-agent-per-room with configurable weights, but auto-graduation (high-trust agents skipping approval) is not yet wired into the policy engine. G6–G7: the data model (* `room_agents` *,* `gateway_agent_id` *,* `agent_to_agent` *direction,* `parent_check_in_id` *), API endpoints, and gateway WebSocket connection are built. Dashboard UI for roster and task trees is not yet built.*

---

3. Out of Scope

- **Agent execution runtime** — Waitroom coordinates; it does not run agent code. Agents run in OpenClaw.
- **Agent-to-agent direct communication** — Agents can delegate tasks to other agents, but this is *mediated delegation* through rooms — not direct messaging or RPC. All tasks pass through rooms with policies, trust scoring, and audit trails.
- **On-premise deployment** — Cloud-hosted only. Self-hosted/VPC deployment deferred.
- **Paid tier or monetization** — No billing or payment code exists in the codebase.
- **SSO/SAML** — Enterprise auth deferred. Currently Supabase Auth (email/password, OAuth).
- **Python / Go SDKs** — Only TypeScript SDK exists currently.
- **LangChain / CrewAI / AutoGen middleware** — No framework-specific integrations beyond the OpenClaw plugin.

4. User Stories

4.1 Agent Developer

- As an agent developer, I want to add a single MCP server URL to my config so that my agent can check in before sensitive actions without writing custom approval logic.
- As an agent developer, I want my agent to receive structured modification data when a human adjusts its proposed action, so it can adapt execution without a second round-trip.
- As an agent developer, I want configurable timeout behaviors (hold, auto_approve, cancel) so my agent handles unresponsive humans gracefully.
- As an agent developer, I want to register watchers that trigger on specific signal patterns so my agent can react to human decisions autonomously.
- As an agent developer, I want my agent to self-register with the Waitroom API (no human needed upfront) and receive a claim token that a human can later use to adopt the agent into their organization.
- As an agent developer, I want a CLI tool (`wr`) so I can manage rooms, check-ins, agents, and signals from my terminal during development.
- As an agent developer, I want my agent to declare capabilities (e.g., "code-review", "email-drafting") so the platform can route relevant tasks to it.
- As an agent developer, I want my agent to post sub-tasks to other specialized agents, with results flowing back via parent-child linking.
- As an agent developer, I want to define each agent's identity (SOUL.md), workspace, and model choice independently, so specialized agents stay focused.

4.2 Human Approver

- As a human approver, I want to see pending check-ins as scannable cards in a feed so I can process many items quickly.
- As a human approver, I want to modify an agent's proposed action (not just approve/reject) so I can inject judgment without blocking the agent entirely.
- As a human approver, I want push notifications on my phone so I can approve urgent check-ins immediately.
- As a human approver, I want to broadcast a signal (budget approved, strategy changed) and have all subscribed agents pick it up automatically.

- As a human approver, I want to claim self-registered agents into my organization using a claim token.
- As a human approver, I want to see which specialized agent is assigned to each task.
- As a human approver, I want to see parent/child task trees when agents delegate to other agents.
- As a human approver, I want to manually assign a task to a specific agent when automatic matching isn't appropriate.

4.3 Gateway Operator

- As a gateway operator, I want to register multiple specialized agents through a single gateway connection, each with declared capabilities and room assignments.
- As a gateway operator, I want agents to hot-reload (join/leave rooms dynamically) when I update my gateway config.
- As a gateway operator, I want incoming tasks routed to the specific agent session (not the main gateway session), so each agent operates in its own workspace with its own SOUL.
- As a gateway operator, I want agent status (online/busy/offline) reported per-agent so the platform knows which agents are available for task matching.

4.4 Enterprise Admin

- As an enterprise admin, I want to define room policies (auto-approve thresholds, forbidden actions) so governance is enforced automatically across all agents regardless of framework.
- As an enterprise admin, I want a complete audit trail of every agent action, human decision, and policy change for compliance reporting.
- As an enterprise admin, I want to view and manage agent trust scores across the organization so I can understand which agents have earned autonomy and which need oversight.

---

5. System Architecture

5.1 High-Level Architecture

Waitroom is a Turborepo monorepo with pnpm workspaces. One core API serves all integration surfaces (MCP, REST, SDK, CLI). The architecture is designed for real-time coordination with strong consistency guarantees on decisions.

```
graph TB
    subgraph "Agent Surfaces"
        MCP["MCP Server<br/>(17 tools, Streamable HTTP + OAuth)"]
        CLI["CLI (wr)<br/>(12 commands + task subcommands)"]
        SDK["TypeScript SDK"]
        REST["REST API<br/>(any HTTP client)"]
    end

    subgraph "Human Surfaces"
        DASH["Dashboard<br/>(Next.js 15 PWA)"]
        MOB["Mobile App<br/>(Expo React Native)"]
    end

    subgraph "Core"
        API["Hono.js API<br/>(port 3001)"]
        PE["Policy Engine"]
        TE["Trust Engine"]
        SYNC["Agent Sync<br/>(WR ↔ OpenClaw)"]
        NS["Notification Service"]
    end

    subgraph "Real-Time"
        WS["WebSocket"]
        SSE["SSE"]
        REDIS["Redis Pub/Sub<br/>(Upstash)"]
        SUPA_RT["Supabase Realtime"]
    end

    subgraph "Data"
        PG["PostgreSQL<br/>(Supabase)"]
        AUTH["Supabase Auth"]
    end

    MCP --> API
    CLI --> API
    SDK --> API
    REST --> API
    DASH --> API
    DASH --> SUPA_RT
    MOB --> API
    MOB --> SUPA_RT

    API --> PE
    API --> TE
    API --> SYNC
    API --> NS
    API --> PG
    API --> AUTH
    API --> WS
    API --> SSE
```

```
    API --> REDIS

    REDIS --> WS
    SUPA_RT --> PG

    style API fill:#0F3460,color:#fff
    style PG fill:#1A1A2E,color:#fff
    style REDIS fill:#E94560,color:#fff
```

5.2 Component Inventory

Component	Type	Technology	Purpose
`@waitroom/api`	App	Hono.js (Node.js, port 3001)	REST API server
`@waitroom/dashboard`	App	Next.js 15 (port 3000)	Human web UI (PWA)
`@waitroom/mobile`	App	Expo React Native	iOS/Android mobile app
`@waitroom/mcp-server`	App	MCP SDK (Node.js, port 3002)	Agent MCP integration (20 tools)
`@waitroom-io/cli`	App	tsup (ESM, node20)	Terminal CLI binary (`wr`)
`@waitroom/shared`	Package	TypeScript	Types, schemas, errors, utils, constants
`@waitroom/db`	Package	Supabase	DB clients + SQL migrations
`@waitroom/sdk-ts`	Package	TypeScript	SDK for API consumption

5.3 Dependency Graph

```
graph BT
    SHARED["@waitroom/shared<br/><i>Types, schemas, errors, utils, constants</
i>"]

    DB["@waitroom/db<br/><i>Supabase clients + SQL migrations</i>"]
    SDK["@waitroom/sdk-ts<br/><i>TypeScript SDK</i>"]

    API["@waitroom/api<br/><i>Hono server</i>"]
    DASH["@waitroom/dashboard<br/><i>Next.js 15</i>"]
    MCP["@waitroom/mcp-server<br/><i>MCP server</i>"]
    CLI_PKG["@waitroom-io/cli<br/><i>wr CLI</i>"]
    MOBILE["@waitroom/mobile<br/><i>Expo app</i>"]

    DB --> SHARED
    SDK --> SHARED

    API --> SHARED
    API --> DB
    DASH --> SHARED
    MCP --> SHARED
    CLI_PKG --> SDK
    CLI_PKG --> SHARED
    MOBILE --> SHARED

    style SHARED fill:#0F3460,color:#fff
    style API fill:#E94560,color:#fff
```

All workspace deps use `workspace:*`. The `@waitroom/shared` package is imported by every other package — changes there cascade everywhere.

5.4 Tech Stack Decision Rationale

- **TypeScript (Hono) for API:** Hono runs on Cloudflare Workers, Deno, Bun, and Node.js. Sub-millisecond cold starts on edge. The MCP SDK reference implementation is TypeScript, so the MCP server and core API share the same language and types.
- **PostgreSQL (Supabase) for primary storage:** ACID transactions for decision integrity (a check-in must be approved exactly once). JSONB columns for flexible policy definitions and signal payloads. Row-level security for multi-tenant isolation. Supabase provides auth, Realtime, and managed Postgres.
- **Redis (Upstash) for real-time state:** Redis Pub/Sub for WebSocket fan-out (human approves → all connected agents receive instantly). Rate limiting via sliding window. Graceful degradation if not configured.

- **Next.js 15 + Tailwind for dashboard:** Server-side rendering for fast initial load. PWA capabilities (service workers, push notifications, install-to-homescreen). Custom `wr-*` color tokens with warm earth tones (bg: #FAF9F7, accent: #C96442).
- **Expo React Native for mobile:** Cross-platform native app. Expo Push for notifications. Supabase Realtime for live updates. Shares types from `@waitroom/shared`.
- **tsup for CLI:** ESM build targeting node20. Workspace deps (`@waitroom/sdk`, `@waitroom/shared`) bundled via `noExternal`. Binary published as `@waitroom-io/cli`.

5.5 Database Schema

PostgreSQL via Supabase. 15 SQL migrations. Row-Level Security on all tables, org-scoped via `auth_org_id()` helper function. Migration 010 adds bidirectional check-ins with threaded messages and multi-agent participation. Migration 013 adds the private `attachments` Supabase Storage bucket. Migration 014 adds sub-agent hierarchy, `room_agents` table, `agent_to_agent` direction, and agent policies per room. Migration 015 seeds 5 starter rooms per org.

```
erDiagram
    organizations ||--o{ users : "has"
    organizations ||--o{ agents : "has"
    organizations ||--o{ rooms : "has"

    agents ||--o{ agent_keys : "has"
    agents ||--o{ check_ins : "creates"
    agents ||--o{ watchers : "creates"
    agents ||--o{ trust_scores : "has"
    agents ||--o{ room_agents : "joins"
    agents ||--o{ check_ins : "assigned to"

    rooms ||--o{ room_members : "has"
    rooms ||--o{ room_agents : "has"
    rooms ||--o{ check_ins : "contains"
    rooms ||--o{ signals : "contains"
    rooms ||--o{ watchers : "contains"
    rooms ||--o{ trust_scores : "scoped to"

    check_ins ||--o{ check_ins : "parent/child"

    users ||--o{ room_members : "joins"
    users ||--o{ check_ins : "decides"
    users ||--o{ push_subscriptions : "has"
    users ||--o{ expo_push_tokens : "has"

    organizations {
        uuid id PK
        text name
        text slug
    }
    users {
        uuid id PK
        uuid org_id FK
        text email
        enum role "owner/admin/member"
    }
    agents {
        uuid id PK
        uuid org_id FK "nullable"
        text name
        text platform
        text claim_token "nullable"
        timestamp claimed_at "nullable"
        uuid gateway_agent_id FK "self-ref, nullable"
        text_array capabilities "default empty"
        text soul_summary "nullable"
        jsonb config "default empty"
        text sync_status "pending/ok/error"
    }
```

```
    agent_keys {
        uuid id PK
        uuid agent_id FK
        text key_prefix "indexed"
        text key_hash
        jsonb scopes
    }
    rooms {
        uuid id PK
        uuid org_id FK
        text name
        text slug "unique per org"
        text description "nullable"
        text agent_instructions "nullable"
        jsonb policies
        jsonb agent_policies "join_mode/max_agents/required_capabilities"
        boolean is_default
    }
    check_ins {
        text id PK "ci_*"
        uuid room_id FK
        uuid agent_id FK
        enum direction "agent_to_human/human_to_agent/agent_to_agent"
        enum status "pending/approved/rejected/modified/expired/withdrawn/
in_progress/submitted"
        enum risk_level "low/medium/high/critical"
        enum urgency_level "low/normal/high/urgent"
        jsonb action
        text context
        jsonb result "nullable"
        jsonb modifications "nullable"
        uuid decided_by FK "nullable"
        uuid created_by FK "nullable"
        uuid claimed_by FK "nullable"
        timestamp claimed_at "nullable"
        boolean help_requested "default false"
        integer message_count "default 0"
        uuid assigned_agent FK "nullable"
        text parent_check_in_id FK "nullable"
        uuid created_by_agent FK "nullable"
    }
    check_in_messages {
        text id PK "cim_*"
        text check_in_id FK
        uuid sender_agent_id FK "nullable"
        uuid sender_user_id FK "nullable"
        enum message_type "update/question/answer/result/system"
        text content
        jsonb metadata "nullable"
    }
    check_in_participants {
```

```
        uuid id PK
        text check_in_id FK
        uuid agent_id FK
        enum role "primary/helper"
        timestamp joined_at
    }
    signals {
        text id PK "sig_*"
        uuid room_id FK
        uuid sent_by FK
        text type
        jsonb payload
    }
    watchers {
        text id PK "w_*"
        uuid room_id FK
        uuid agent_id FK
        text_array event_types
        text webhook_url
    }
    trust_scores {
        uuid agent_id FK
        uuid room_id FK
        float score "0-100"
        jsonb counters
    }
    room_agents {
        uuid id PK
        uuid room_id FK
        uuid agent_id FK
        uuid org_id FK
        text_array capabilities "default empty"
        enum status "online/busy/offline/pending_approval"
        timestamp joined_at
        timestamp updated_at
    }
```

5.6 Prefixed IDs

All entities use self-describing prefixed text IDs generated via `generateId(prefix, length)` in `@waitroom/shared`:

Prefix	Entity	Example
`wr_`	Agent API keys	`wr_a8k2m9x...`
`wr_claim_`	Claim tokens (7-day expiry)	`wr_claim_7x2k...`

Prefix	Entity	Example
`ci_`	Check-ins	`ci_8x7k2m...`
`cim_`	Check-in messages	`cim_4n2x8k...`
`ra_`	Room agent memberships	`ra_7k3m9x...`
`sig_`	Signals	`sig_3m8x2k...`
`w_`	Watchers	`w_9k2x8m...`
`rm_`	Rooms	`rm_4x7k2m...`

5.7 Enums

Enum	Values
`risk_level`	low, medium, high, critical
`urgency_level`	low, normal, high, urgent
`check_in_status`	pending, approved, rejected, modified, expired, withdrawn, in_progress, submitted
`check_in_direction`	agent_to_human, human_to_agent, agent_to_agent
`message_type`	update, question, answer, result, system
`participant_role`	primary, helper
`timeout_action`	auto_approve, cancel, hold
`actor_type`	agent, human, system
`user_role`	owner, admin, member
`agent_room_status`	online, busy, offline, pending_approval
`policy_action`	auto_approve, require_approval, forbid

6. Gateway & Sub-Agent Architecture

This section describes the multi-agent model. A single gateway agent connects to Waitroom and manages multiple specialized sub-agents, each with their own capabilities, SOUL, and room assignments.

6.1 Core Model

Three foundational principles:

1. **Waitroom is the control plane.** Agents are created, configured, and edited from the Waitroom UI. OpenClaw is the execution layer. Waitroom tells OpenClaw what agents to create; OpenClaw pushes back SOUL, capabilities, and status to keep Waitroom in sync.
2. **The gateway agent is the orchestrator.** There is no platform-side matching engine. When a task is posted to a room, the gateway agent receives it, evaluates room context and available sub-agents, picks a lead, and delegates. Orchestration logic lives in agent behavior, not platform code.
3. **Rooms drive agent membership.** When a room is created, Waitroom notifies the gateway. All existing agents evaluate whether they should request joining. Admins see all available agents and can force-add or approve join requests.

6.2 Gateway Registration

The gateway (e.g., an OpenClaw instance) reads `api.waitroom.io/skill.md` which contains self-register instructions. This establishes the trust anchor:

```
sequenceDiagram
    participant OC as OpenClaw Gateway
    participant API as Waitroom API
    participant Human as Human (Dashboard)

    OC->>API: Reads /skill.md → follows self-register instructions
    OC->>API: POST /v1/agents/self-register<br/>{name: "Gopi's Gateway",
platform: "openclaw"}
    API-->>OC: { api_key: "wr_...", claim_token: "wr_claim_..." }
    OC-->>Human: Shares claim_token
    Human->>API: POST /v1/agents/claim {claim_token}
    Note over API: Gateway now belongs to org
```

The gateway agent is the org's single trust anchor.

6.3 Agent Creation (Bidirectional Sync)

Agents can be created from **two directions**, and Waitroom stays in sync regardless:

6.3.1 Human creates agent from Waitroom UI → Runtime

```
sequenceDiagram
    participant Admin as Human (Dashboard)
    participant WR as Waitroom API
    participant GW as OpenClaw Gateway

    Admin->>WR: "Create Agent: Creative Writer"<br/>{name, capabilities, rooms,
model}
    WR->>WR: Creates agent record (pending sync)
    WR->>GW: POST /agents/create (webhook/WS)<br/>{id, name, capabilities, rooms,
model}
    GW->>GW: Creates agent dir, SOUL.md, workspace
    GW-->>WR: POST /v1/agents/{id}/sync<br/>{soulSummary, capabilities, status:
"online"}
    Note over WR: Agent record updated with SOUL
    WR-->>Admin: Agent created + synced
```

The human defines the high-level intent (name, capabilities, rooms, model) from the Waitroom dashboard. OpenClaw handles the execution-side setup (directory structure, SOUL.md generation, workspace). OpenClaw then pushes the finalized SOUL and capabilities back so Waitroom is always in sync.

6.3.2 Agent defined in runtime config → pushed to Waitroom

```
sequenceDiagram
    participant GW as OpenClaw Gateway
    participant API as Waitroom API

    Note over GW: Boot / config reload
    GW->>GW: Reads agents.list from config
    loop For each agent
        GW->>API: POST /v1/agents/sync<br/>{gatewayAgentId, name,
capabilities,<br/>soulSummary, rooms, status}
        API-->>GW: { agentId: "uuid", rooms: [{status}] }
    end
```

Either direction works. Waitroom is always the source of truth for *what agents exist and what rooms they're in*. OpenClaw is the source of truth for *how agents behave* (SOUL, workspace, skills).

6.3.3 Agent Editing from Waitroom UI

From the Waitroom dashboard, admins can: - **Edit agent properties directly** — name, capabilities, room assignments - **Chat with the agent's behavior** — interactive session to refine the SOUL (e.g., "Be more concise in your blog posts", "Always include SEO metadata"). Changes push to OpenClaw which regenerates the SOUL.md. - **View/edit SOUL summary** — the natural language description visible to humans and other agents

All edits sync bidirectionally: Waitroom updates its records, then pushes changes to OpenClaw via webhook/WebSocket, OpenClaw confirms.

6.4 Agent Definition (Runtime Side)

Each agent has its own identity, workspace, filtered skills, and model choice:

```
{
  agents: {
    list: [
      {
        id: "creative-writer",
        name: "Creative Writer",
        workspace: "~/.openclaw/agents/creative-writer/workspace",
        agentDir: "~/.openclaw/agents/creative-writer/agent",
        skills: ["summarize", "web-search"],
        model: { primary: "anthropic/claude-sonnet-4-20250514" },
        waitroom: {
          capabilities: ["blog-posts", "social-copy", "essays", "editing"],
          autoJoin: true
        }
      },
      {
        id: "illustrator",
        name: "Illustrator",
        workspace: "~/.openclaw/agents/illustrator/workspace",
        agentDir: "~/.openclaw/agents/illustrator/agent",
        skills: ["nano-banana-pro"],
        model: { primary: "anthropic/claude-sonnet-4-20250514" },
        waitroom: {
          capabilities: ["illustrations", "hero-images", "diagrams", "social-
graphics"],
          autoJoin: true
        }
      },
      {
        id: "code-reviewer",
        name: "Code Reviewer",
        workspace: "~/.openclaw/agents/code-reviewer/workspace",
        agentDir: "~/.openclaw/agents/code-reviewer/agent",
        skills: ["github", "coding-agent"],
        model: { primary: "anthropic/claude-opus-4-6" },
        waitroom: {
          capabilities: ["code-review", "pr-review", "refactoring"],
          autoJoin: true
        }
      }
    ]
  }
}
```

Each agent has its own isolated filesystem:

```
~/.openclaw/agents/creative-writer/
  ├── agent/
  │   ├── AGENTS.md      # behavior instructions
  │   ├── SOUL.md        # identity + specialization
  │   └── TOOLS.md       # local notes
  └── workspace/         # working files, drafts, outputs
```

> *Note:* `waitroom.rooms` *is no longer in the agent config. Room membership is managed from the Waitroom side (see 6.5). Agents declare capabilities; rooms pull agents in.*

6.5 Room-Driven Agent Membership

Room creation triggers agent awareness. This is the reverse of the gateway-pushes-agents model:

```
sequenceDiagram
    participant Admin as Human (Dashboard)
    participant WR as Waitroom API
    participant GW as OpenClaw Gateway

    Admin->>WR: Create room "Content"<br/>{agent_policy: {join: "approval"}}
    WR->>GW: room.created event<br/>{roomId, name, slug, requiredCapabilities}

    Note over GW: Each agent evaluates:<br/>"Do my capabilities match this room?"

    GW->>WR: POST /v1/rooms/content/agents/join<br/>{agentId: "creative-writer",
capabilities: [...]}
    GW->>WR: POST /v1/rooms/content/agents/join<br/>{agentId: "illustrator",
capabilities: [...]}

    Note over WR: Room admin sees join requests
    Admin->>WR: Approve creative-writer ✅
    Admin->>WR: Approve illustrator ✅
```

Room agent roster view: The dashboard room view shows: - **Joined agents** — approved members with status (online/busy/offline) - **Pending agents** — awaiting admin approval - **All available agents** — every agent in the org, even those not in this room, so admin can force-add

Force-add: Admin can add any org agent to any room, regardless of join policy. This bypasses the agent's self-evaluation.

6.6 Task Orchestration (Main Agent as Router)

When a human posts a task to a room, the platform does **not** run a matching algorithm. Instead:

```
sequenceDiagram
    participant H as Human
    participant WR as Waitroom API
    participant GW as Gateway (Main Agent)
    participant LA as Lead Agent
    participant HA as Helper Agent

    H->>WR: Post task to "Content" room<br/>"Write a blog post about AI safety
with illustrations"
    WR->>GW: Task notification<br/>{task, room context, agent roster}

    Note over GW: Main agent evaluates:<br/>- Room context (instructions,
policies)<br/>- Available agents + capabilities<br/>- Task requirements
    GW->>GW: Picks lead: Creative Writer<br/>(best fit for primary intent)

    GW->>WR: POST /v1/check-ins/{id}/claim<br/>(on behalf of Creative Writer)
    Note over WR: status → in_progress,<br/>assigned_agent: creative-writer

    GW->>LA: Inject task into Creative Writer session

    LA->>LA: Writes blog post
    LA->>LA: "I need illustrations"
    LA->>GW: Request sub-task for Illustrator
    GW->>WR: POST /v1/rooms/content/check-ins<br/>{parentCheckInId, originAgent:
"creative-writer"}
    GW->>WR: POST /v1/check-ins/{child}/claim<br/>(on behalf of Illustrator)
    GW->>HA: Inject sub-task into Illustrator session

    HA->>HA: Creates illustrations
    HA->>GW: Submit result
    GW->>WR: POST /v1/check-ins/{child}/submit

    Note over GW: Child complete → notify Lead
    GW->>LA: Child task result (images)

    LA->>LA: Assembles post + images
    LA->>GW: Submit final result
    GW->>WR: POST /v1/check-ins/{id}/submit
    WR->>H: Result ready for review
```

Key properties: - **No matching engine in the platform.** The main agent IS the router. It has the room context, the agent roster, and the intelligence to make the right call. - **Orchestration lives in the main agent's behavior.** The main agent's SOUL/instructions tell it how to evaluate tasks, when to pick leads, when to involve helpers, and when to post check-ins for human clarification. - **The**

lead agent coordinates. Once claimed, the lead agent owns the task. It can delegate sub-tasks to other agents in the room (or other rooms). If the lead is uncertain, it posts a check-in so humans can clarify. - **Same primitives.** Everything uses existing check-ins, claims, messages, and submit. No new coordination mechanisms needed — just smarter agents.

6.7 Agent-to-Agent Delegation

The lead agent can delegate sub-tasks. This uses parent-child linking on check-ins:

How it works: 1. Lead agent determines it needs help (e.g., writer needs illustrations) 2. Lead agent tells the gateway to post a sub-task with `parentCheckInId` linking to the original task 3. Gateway posts the sub-task to the appropriate room 4. Main agent picks the best agent for the sub-task and claims on its behalf 5. Sub-task result flows back to lead agent via parent link notification

Guardrails: - **Max chain depth:** 5 levels (enforced at creation: `depth = parent.depth + 1`) - **Max children per task:** 10 (configurable per room) - **Per-agent rate limit:** Existing 60 req/min applies to agent-posted tasks - **Human check-ins for doubt:** When the lead agent is uncertain about anything, it posts a check-in for human clarification rather than guessing

6.8 Agent Lifecycle

```
stateDiagram-v2
    [*] --> created: Human creates from UI / defined in config
    created --> syncing: Bidirectional sync (WR ↔ OpenClaw)
    syncing --> available: Agent exists in org, no rooms yet
    available --> join_requested: Room created → agent requests join
    available --> force_added: Admin force-adds to room
    join_requested --> joined: Admin approves
    join_requested --> rejected: Admin rejects
    force_added --> joined
    joined --> online: Gateway reports ready
    online --> busy: Working a task
    busy --> online: Task complete
    online --> offline: Gateway disconnects
    offline --> online: Gateway reconnects

    note right of available: Visible in "All Agents" roster
    note right of joined: Visible in room agent roster
    note right of busy: Lead agent coordinates
```

Status transitions: - **online** — Ready for tasks. Available for main agent to select. - **busy** — Currently working a task. Main agent can still select if critical. - **offline** — Gateway disconnected or agent removed.

6.9 Room Agent Policy

Rooms control how agents join via `agent_policy` JSONB:

```
{
  "join": "auto",
  "maxAgents": 10,
  "requiredCapabilities": ["writing"]
}
```

Policy	Behavior
`auto`	Any org agent matching required capabilities joins immediately
`approval`	Admin must approve each agent join request
`invite_only`	Admin explicitly adds agents; self-join requests rejected

`maxAgents` — Optional limit on agents per room. Join rejected if at capacity. `requiredCapabilities` — Optional filter. Agent must declare at least one matching capability to join. **Admin override** — Admin can always force-add any org agent, regardless of policy.

6.10 Bidirectional Sync Protocol

Waitroom and OpenClaw communicate via a defined sync protocol over WebSocket:

Direction	Event	Trigger	Payload
WR → OpenClaw	`agent.sync` (create)	Human creates agent from UI	`{name, capabilities, model, rooms}`
WR → OpenClaw	`agent.sync` (update)	Human edits agent from UI	`{changes}`
WR → OpenClaw	`agent.sync` (delete)	Human deletes agent from UI	`{agentId}`
WR → OpenClaw	`agent.task.assigned`	Task posted to room	`{task, roomContext, agentRoster, suggested_lead}`
	`task.decision`		

Direction	Event	Trigger	Payload
WR → OpenClaw		Human approves/rejects result	`{checkInId, decision, reason}`
OpenClaw → WR	`agent_sync_ack`	Agent acks create/update/delete	`{agentId, action, status}`
OpenClaw → WR	`agent_status`	Agent status change	`{agentId, status}`

Transport: WebSocket (primary, already exists) or webhook callbacks (fallback).

6.11 Child Task Rejection

When a child task result is rejected: 1. Rejection is sent back to the same agent with the rejection reason 2. Agent can revise and resubmit, or release the task back to the pool 3. Parent agent remains in `in_progress` state, waiting for child resolution 4. Same mechanics as existing task rejection — no new primitives needed

---

7. API Design

Hono.js REST API. All `/v1/*` routes require authentication (agent API key or human JWT). Three route guards: `agentOnly()`, `humanOnly()`, `claimedAgentOnly()` (agent auth + `org_id` not null).

7.1 Agent Endpoints

Method	Endpoint	Auth	Description
POST	`/v1/agents/register`	humanOnly	Human creates agent, receives API key
POST	`/v1/agents/self-register`	public (IP-limited)	Agent self-registers, gets key + claim token
POST	`/v1/agents/claim`	humanOnly	Claim self-registered agent into org

Method	Endpoint	Auth	Description
GET	`/v1/agents/me`	agentOnly	Agent introspection (status, claim token)
GET	`/v1/agents`	humanOnly	List org agents with pagination
GET	`/v1/agents/:agent`	humanOnly	Get single agent details
PATCH	`/v1/agents/:agent`	humanOnly	Update agent name/description/ platform
DELETE	`/v1/agents/:agent`	humanOnly	Delete agent

7.1.1 Sub-Agent Management Endpoints

Method	Endpoint	Auth	Description
POST	`/v1/agents/sync`	claimedAgentOnly	Gateway pushes agent definition (SOUL, capabilities, status) to Waitroom
POST	`/v1/agents/create`	humanOnly	Human creates agent from dashboard; triggers OpenClaw webhook
PATCH	`/v1/agents/:agent/soul`	humanOnly	Update agent SOUL/behavior from dashboard; syncs to OpenClaw
PATCH	`/v1/agents/:agent/status`	claimedAgentOnly	Update agent status (online/busy/ offline)
POST	`/v1/rooms/:room/agents/join`	claimedAgentOnly	Agent requests to join room with capabilities
POST	`/v1/rooms/:room/agents/leave`	claimedAgentOnly	Remove agent from room roster
POST	`/v1/rooms/:room/agents/force-add`	humanOnly	Admin force-adds any org agent to room
POST		humanOnly	

Method	Endpoint	Auth	Description
	`/v1/rooms/:room/agents/:agent/approve`		Admin approves pending join request
GET	`/v1/rooms/:room/agents`	any auth	List room agent roster (joined, pending, available)

Sync request body (OpenClaw → Waitroom):

```
{
  "gatewayAgentId": "creative-writer",
  "name": "Creative Writer",
  "capabilities": ["blog-posts", "social-copy", "essays", "editing"],
  "soulSummary": "I write clear, engaging content. No code, no ops, just words.",
  "status": "online"
}
```

Create request body (Waitroom UI → OpenClaw → Waitroom):

```
{
  "name": "Creative Writer",
  "capabilities": ["blog-posts", "social-copy", "essays", "editing"],
  "model": "anthropic/claude-sonnet-4-20250514",
  "rooms": ["content", "marketing"]
}
```

7.2 Check-in Endpoints (Agent → Human)

Method	Endpoint	Auth	Description
POST	`/v1/rooms/:room/check-in`	claimedAgentOnly	Create check-in (triggers policy engine). Supports `parentCheckInId` for task chaining.
GET	`/v1/check-ins/:id/status`	any auth	Poll for decision status
DELETE	`/v1/check-ins/:id`	claimedAgentOnly	Withdraw pending check-in
GET		humanOnly	List pending check-ins (filterable)

Method	Endpoint	Auth	Description
	`/v1/rooms/:room/pending`		
GET	`/v1/check-ins`	any auth	List check-ins across all rooms (filterable by direction, status)
POST	`/v1/check-ins/:id/approve`	humanOnly	Approve with optional reason
POST	`/v1/check-ins/:id/reject`	humanOnly	Reject with reason
POST	`/v1/check-ins/:id/modify`	humanOnly	Approve with structured modifications

7.2.1 Task Endpoints (Human → Agent — Bidirectional)

Method	Endpoint	Auth	Description
POST	`/v1/rooms/:room/tasks`	humanOnly	Post task for agents to claim
GET	`/v1/rooms/:room/tasks`	any auth	List tasks in room
POST	`/v1/check-ins/:id/claim`	claimedAgentOnly	Claim a task (status → in_progress)
POST	`/v1/check-ins/:id/release`	claimedAgentOnly	Release claimed task back to pool
POST	`/v1/check-ins/:id/help`	claimedAgentOnly	Flag task as needing help
POST	`/v1/check-ins/:id/join`	claimedAgentOnly	Join as helper agent
POST	`/v1/check-ins/:id/submit`	claimedAgentOnly	Submit result (status → submitted)
POST	`/v1/check-ins/:id/messages`	any auth	Post threaded message
GET	`/v1/check-ins/:id/messages`	any auth	List thread messages

Method	Endpoint	Auth	Description
GET	`/v1/check-ins/claimed`	agentOnly	List tasks claimed by calling agent

7.3 Room, Signal, Watcher, and Audit Endpoints

Method	Endpoint	Auth	Description
POST	`/v1/rooms`	humanOnly	Create room with optional policies
GET	`/v1/rooms`	any auth	List org rooms
GET	`/v1/rooms/:room`	any auth	Get room by slug or ID
PATCH	`/v1/rooms/:room`	humanOnly	Update name/description
DELETE	`/v1/rooms/:room`	humanOnly	Delete room
PUT	`/v1/rooms/:room/policies`	humanOnly	Update room policies
POST	`/v1/rooms/:room/signal`	humanOnly	Broadcast signal to watchers
POST	`/v1/rooms/:room/watch`	claimedAgentOnly	Create watcher subscription
DELETE	`/v1/watchers/:id`	claimedAgentOnly	Remove watcher
GET	`/v1/rooms/:room/audit`	humanOnly	Query audit log

7.4 Skill/Discovery Endpoints

Served at top-level paths (not under `/v1/`):

Path	Description
`GET /skill.md`	Markdown skill file with self-register instructions, check-in guide, API reference
`GET /skill.json`	Machine-readable metadata (triggers, endpoints, auth)

Path	Description
`GET /heartbeat.md`	Heartbeat routine (poll `/v1/home`, handle signals)
`GET /rules.md`	Agent behavioral principles

7.5 Real-Time Channels

Four transport layers, all gracefully degrading if Redis/Supabase not configured:

```
graph LR
    subgraph "Human Decision"
        H["Human approves<br/>check-in"]
    end

    subgraph "API Server"
        API["Hono API"]
        PG["PostgreSQL<br/>(write)"]
        RP["Redis Pub/Sub<br/>(publish)"]
        SR["Supabase Realtime<br/>(postgres_changes)"]
    end

    subgraph "Agent Transports"
        WS["WebSocket<br/>(?token=wr_*)"]
        SSE["SSE<br/>(/v1/rooms/:room/events)"]
        RPOLL["Redis Subscribe<br/>(room:{id}:events)"]
    end

    subgraph "Human Transports"
        DASH["Dashboard<br/>(useRealtimeCheckIns)"]
        MOB_RT["Mobile App<br/>(Supabase Realtime)"]
    end

    H --> API
    API --> PG
    API --> RP
    PG --> SR

    RP --> WS
    RP --> RPOLL
    SR --> SSE
    SR --> DASH
    SR --> MOB_RT

    style H fill:#2E7D32,color:#fff
    style API fill:#0F3460,color:#fff
    style RP fill:#E94560,color:#fff
```

- **Redis Pub/Sub:** For agent consumers. Channels: `room:{id}:events`, `checkin:{id}`. Backed by Upstash Redis.
- **SSE:** `GET /v1/rooms/:room/events`. Server-sent events backed by Supabase Realtime `postgres_changes`.
- **WebSocket:** Auth via `?token=wr_...` query param. Subscribe/unsubscribe to rooms via JSON messages. Backed by Supabase Realtime channels.
- **Supabase Realtime (direct):** Dashboard uses `useRealtimeCheckIns` hook subscribing to `postgres_changes` directly.

Gateway WebSocket Events:

Event	Payload	Description
`agent.task.assigned`	`{ checkInId, roomId, agentId }`	Task routed to this agent
`agent.task.child.completed`	`{ childCheckInId, parentCheckInId, result }`	A child task posted by this agent has completed
`agent.roster.updated`	`{ roomId, agents[] }`	Room agent list changed (join/leave/status)

7.6 MCP Server Tools

The MCP server exposes **27 tools** with per-session `McpServer` instances. Primary transport: Streamable HTTP at `POST /` / `GET /` / `DELETE /` with session management via `mcp-session-id` header. OAuth mode available when `MCP_SERVER_URL` is set (Supabase auth + OAuth flow). Legacy SSE transport at `GET /sse` + `POST /messages?sessionId=`.

Core Tools (Original 7):

Tool Name	Description
`waitroom_setup`	Self-register or check connection status; polls for pending tasks on connect
`waitroom_check_in`	Submit a check-in to request human approval
`waitroom_check_status`	Poll for decision (approved/rejected/modified)
`waitroom_watch`	Subscribe a watcher with trigger conditions
`waitroom_signal`	Broadcast a decision signal to watching agents
`waitroom_list_rooms`	List available rooms and their current status
`waitroom_withdraw`	Withdraw a pending check-in

Task Tools (New — Bidirectional):

Tool Name	Description
`waitroom_claim_task`	Claim a human-posted task (status → in_progress)
`waitroom_release_task`	Release a claimed task back to the pool
`waitroom_request_help`	Flag a task as needing help from other agents
`waitroom_post_message`	Post a threaded message on a check-in/task
`waitroom_list_check_ins`	List check-ins across all rooms with filters
`waitroom_list_claimed`	List tasks claimed by this agent
`waitroom_list_room_tasks`	List tasks in a specific room
`waitroom_get_thread`	Get full message thread for a check-in/task

Context Tools (New):

Tool Name	Description
`waitroom_read_signals`	Read recent signals broadcast to a room
`waitroom_read_rules`	Read room-specific agent instructions and policies

Multi-Agent Tools (Planned):

Tool Name	Description
`waitroom_sync_agent`	Push agent definition (SOUL, capabilities, status) to Waitroom
`waitroom_join_room`	Request to join a room with capability declaration
`waitroom_post_subtask`	Post a child task linked to current parent (agent-to-agent delegation)
`waitroom_room_roster`	List agents in a room with status and capabilities

8. Policy Engine and Trust Scoring

8.1 Policy Engine

Room policies are stored as JSONB on the `rooms` table, validated by `roomPoliciesSchema` (Zod). The policy engine in `policy.service.ts` evaluates in strict priority order:

```
flowchart TD
    CI["Check-in Created"] --> F{"Forbid rules<br/>match?"}
    F -->|Yes| REJ["❌ REJECTED"]
    F -->|No| AA{"Auto-approve<br/>rules match?"}
    AA -->|Yes| APP["✅ APPROVED"]
    AA -->|No| TR{"Trust score<br/>above threshold?"}
    TR -->|Yes| APP
    TR -->|No| DEF{"Default action"}
    DEF -->|require_approval| PEND["⏳ PENDING<br/>(human review)"]
    DEF -->|auto_approve| APP
    DEF -->|forbid| REJ

    style REJ fill:#C62828,color:#fff
    style APP fill:#2E7D32,color:#fff
    style PEND fill:#F57F17,color:#fff
```

1. **Forbid rules** (highest priority) — if any match, check-in is immediately rejected
2. **Auto-approve rules** — if any match, check-in is approved without human review
3. **Trust-based thresholds** — per risk level (low/medium have separate score thresholds)
4. **Default action** (fallback) — usually `require_approval`

Policy conditions can match on: `risk_level`, `action_type` (via `action.type` field), `urgency_level`, and trust_score thresholds.

8.2 Trust Scoring

Per-agent, per-room trust scores maintained in `trust.service.ts`:

Parameter	Value	Notes
Initial Score	15	Starting trust for new agent-room pairs
Range	0 – 100	Clamped at boundaries
Approval weight	+1.0	Each approval increases score

Parameter	Value	Notes
Modification weight	+0.6	Partial credit for modified approvals
Rejection weight	-0.3	Rejections reduce trust
Expiry weight	-0.1	Timeouts slightly reduce trust

```
graph LR
    NEW["New Agent<br/>Score: 15"] -->|"Approvals (+1.0)"| MED["Building
Trust<br/>Score: 40-60"]
    MED -->|"Consistent approvals"| HIGH["Trusted<br/>Score: 80+"]
    HIGH -->|"Rejections (-0.3)"| MED
    MED -->|"Rejections (-0.3)"| LOW["Low Trust<br/>Score: 0-15"]
    LOW -->|"Approvals (+1.0)"| MED

    style NEW fill:#F57F17,color:#fff
    style MED fill:#0F3460,color:#fff
    style HIGH fill:#2E7D32,color:#fff
    style LOW fill:#C62828,color:#fff
```

8.3 Check-in Lifecycle (Agent → Human)

```
stateDiagram-v2
    [*] --> pending: Agent creates check-in
    pending --> approved: Human approves
    pending --> rejected: Human rejects
    pending --> modified: Human modifies
    pending --> expired: Timeout triggers
    pending --> withdrawn: Agent withdraws

    approved --> [*]
    rejected --> [*]
    modified --> [*]
    expired --> [*]
    withdrawn --> [*]

    note right of pending: Policy engine may auto-resolve\n(auto_approve or
forbid)
    note right of modified: Includes structured\nmodification payload
```

8.4 Task Lifecycle (Human → Agent — Bidirectional)

```
stateDiagram-v2
    [*] --> pending: Human posts task
    pending --> in_progress: Agent claims task
    in_progress --> submitted: Agent submits result
    in_progress --> pending: Agent releases task
    submitted --> approved: Human approves result
    submitted --> rejected: Human rejects result
    pending --> expired: Timeout triggers
    pending --> withdrawn: Human withdraws

    approved --> [*]
    rejected --> [*]
    expired --> [*]
    withdrawn --> [*]

    note right of in_progress: Agent can request help\n(help_requested=true)
    note right of in_progress: Other agents can join\nas helpers via /join
    note right of submitted: Threaded messages\nthroughout lifecycle
```

8.5 Threaded Messages

Every check-in (both directions) supports a message thread. Messages are typed (`update`, `question`, `answer`, `result`, `system`) and can be posted by agents or humans. The `message_count` on the check-in is incremented atomically.

8.6 Multi-Agent Participation

Tasks support multiple agents via `check_in_participants`. The claiming agent is the `primary` participant (trust weight 1.0x). Agents that join via `/join` are `helper` participants (trust weight 0.5x). Trust scoring applies to all participants when a decision is made.

---

9. Requirements

9.1 Must-Have (P0) — Launch

#	Requirement	Status
1	Core API: Rooms CRUD, check-in lifecycle (create → pending → approved/rejected/modified/expired/withdrawn), signal broadcast, watcher registration	✅ Built
2	Three-decision model: Approve, Reject, and Modify with structured modification payloads that agents can parse and act on	✅ Built
3	MCP Server: 7 tools (waitroom_setup, check_in, check_status, watch, signal, list_rooms, withdraw)	✅ Built
4	OpenClaw Skill: Skill markdown file exists at `GET /skill.md`, but OpenClaw-specific integration not yet implemented	⚠️ Partial
5	Real-time delivery: WebSocket, SSE, Redis Pub/Sub, and Supabase Realtime channels	✅ Built
6	Human dashboard (PWA): Feed-based UI with check-in cards, approve/reject/modify, push notifications, service worker	✅ Built
7	Auth: Dual auth (human JWT via Supabase + agent API keys with salted SHA-256 hash, timing-safe comparison)	✅ Built
8	Default room: Auto-created on signup via `create_default_room()` trigger	✅ Built
9	Basic policies: Forbid rules, auto-approve rules, trust thresholds, default action, evaluated in strict order	✅ Built
10	Audit trail: Immutable append-only log via `logAuditEvent()` on every coordination event	✅ Built
11	Trust scoring: Per-agent-per-room scores with weights. Auto-graduation logic not yet wired into policy evaluation	⚠️ Partial

9.2 Nice-to-Have (P1) — Post-Launch

#	Requirement	Status
1	TypeScript SDK: `@waitroom/sdk-ts` with `client.checkIns.checkInAndWait()` convenience method	✅ Built
2	Python SDK	❌ Not Built
3	Trust-based auto-graduation: Score tracking works but auto-approve bypass not connected	❌ Not Built
4	Escalation chains: Not implemented. Timeout actions limited to auto_approve, cancel, hold	❌ Not Built
5	Batch approval: Not implemented in dashboard or API	❌ Not Built
6	AI-summarized review queues	❌ Not Built
7	Urgency-adaptive notifications: Push notifications work (Web Push + Expo Push), but digest/batching not implemented	⚠️ Partial
8	LangChain/CrewAI/AutoGen middleware integrations	❌ Not Built

9.3 Future Considerations (P2) — Enterprise

1. Portable trust scores across organizations (Trust Network)
2. SSO/SAML authentication for enterprise orgs
3. Role-based access control (RBAC) with granular permissions
4. Compliance policy templates (HIPAA, SOX, GDPR)
5. On-premise / VPC deployment option
6. ~~Native iOS and Android mobile apps~~ — Built via Expo React Native
7. Go SDK
8. Advanced analytics and governance dashboards

---

10. Infrastructure and Deployment

10.1 Current Stack (As Implemented)

Component	Platform	Notes
Core API	Hono.js (Node.js)	Port 3001. `tsx watch` for dev. No edge deployment yet.
Database	Supabase (Postgres)	Managed Postgres with auth, Realtime, RLS. 15 migrations.
Redis	Upstash	Sliding window rate limiting + pub/sub. Graceful skip if not configured.
Dashboard	Next.js 15	PWA with service worker, manifest, Web Push notifications.
Mobile	Expo (React Native)	Native iOS/Android. Expo Push for notifications.
Auth	Supabase Auth	JWT validation. Agent keys managed in-house with salted SHA-256.

10.2 Environment Variables

Each app has its own env file — env vars are **not** shared from root:

- `apps/api/.env` — SUPABASE_URL, SUPABASE_ANON_KEY, SUPABASE_SERVICE_ROLE_KEY, UPSTASH_REDIS_REST_URL, UPSTASH_REDIS_REST_TOKEN, API_PORT, DASHBOARD_URL, VAPID_PUBLIC_KEY, VAPID_PRIVATE_KEY, VAPID_SUBJECT
- `apps/dashboard/.env.local` — NEXT_PUBLIC_API_URL, NEXT_PUBLIC_SUPABASE_URL, NEXT_PUBLIC_SUPABASE_ANON_KEY
- `apps/mcp-server/.env` — API_BASE_URL, WAITROOM_API_KEY, MCP_SERVER_PORT, DASHBOARD_URL. OAuth mode adds: MCP_SERVER_URL, SUPABASE_URL, SUPABASE_ANON_KEY, OAUTH_CLIENT_ID, OAUTH_CLIENT_SECRET

10.3 Rate Limiting

- **Agent writes:** 60 req/min sliding window via Upstash Redis, per `agent.id`. POST/PUT/PATCH/DELETE only.

- **IP-based limit:** On `/v1/agents/self-register` only (separate middleware).
- **Human JWT users:** Not rate limited at application level.
- **Graceful degradation:** Rate limiting skipped entirely if Upstash Redis not configured.

10.4 Scaling Considerations

- **WebSocket connections:** Each waiting agent holds one WebSocket. Redis Pub/Sub handles fan-out across multiple API instances.
- **Check-in throughput:** Target 1,000 check-ins/second. Postgres with proper indexing handles this. Hot path (status polling) can be served from Redis cache.
- **Audit trail volume:** Append-only writes. Partitioning by month recommended for production.

---

11. Mobile Strategy

> ***Original spec:*** *PWA first, native later (Phase 3).* ***Actual implementation:*** *Both PWA and native Expo app built in parallel.*

11.1 PWA (Dashboard)

- **Push notifications:** Web Push API via service worker. VAPID keys configured in API environment.
- **Install-to-homescreen:** PWA manifest configured. Full-screen app-like experience.
- **Responsive design:** Mobile-first card layout with warm earth tone design system (bg: #FAF9F7, accent: #C96442).

11.2 Native App (Expo React Native)

- **Screens:** Check-in feed, room list, room detail, settings, login. Tab-based navigation.
- **Push notifications:** Expo Push Notifications (not Web Push). Separate `expo_push_tokens` table and `/v1/expo-push/token` endpoint.
- **Real-time:** Supabase Realtime subscription in `useRealtimeCheckIns` hook.
- **Auth:** Supabase Auth with secure token storage.
- **Design:** Custom theme with consistent color tokens shared with dashboard.

12. Security Model

12.1 Authentication (Dual Auth)

Every `/v1/*` route passes through auth middleware (`apps/api/src/middleware/auth.ts`) which implements two auth modes:

```
flowchart TD
    REQ["Incoming Request"] --> HDR{"Authorization<br/>header?"}
    HDR -->|"Bearer wr_*"| AGENT["Agent Key Auth"]
    HDR -->|"Bearer jwt..."| HUMAN["Human JWT Auth"]
    HDR -->|"None"| DENY["401 Unauthorized"]

    AGENT --> PREFIX["Lookup by key_prefix<br/>(first 11 chars)"]
    PREFIX --> HASH["SHA-256 + salt<br/>timing-safe compare"]
    HASH -->|Match| CTX_A["Set AuthAgent context<br/>(agent.id, agent.org_id)"]
    HASH -->|No match| DENY

    HUMAN --> SUPA["supabase.auth.getUser()"]
    SUPA -->|Valid| CTX_H["Set AuthUser context<br/>(user.id, user.org_id)"]
    SUPA -->|Invalid| DENY

    style DENY fill:#C62828,color:#fff
    style CTX_A fill:#2E7D32,color:#fff
    style CTX_H fill:#2E7D32,color:#fff
```

- **Agent keys (`wr_*`):** Looked up by first 11 chars (`key_prefix` index), then salted SHA-256 hash verified with `crypto.timingSafeEqual` . Sets `AuthAgent` context.
- **Human JWT:** Supabase JWT validated via `supabase.auth.getUser()` . Sets `AuthUser` context.

12.2 Route Guards

Guard	Requires	Used By
`agentOnly()`	Agent authentication	`GET /v1/agents/me`
`humanOnly()`	Human JWT authentication	All decision endpoints, room/agent management

Guard	Requires	Used By
`claimedAgentOnly()`	Agent auth + `org_id !== null`	Check-in creation, watcher creation

12.3 Agent Self-Registration and Claiming

```
sequenceDiagram
    participant Agent
    participant API
    participant Human
    participant Dashboard

    Agent->>API: POST /v1/agents/self-register<br/>(name, platform)
    Note over API: IP rate limited
    API-->>Agent: { api_key: "wr_...",<br/>claim_token: "wr_claim_..." }

    Agent->>API: GET /v1/agents/me<br/>(wr_* key)
    API-->>Agent: { status: "unclaimed",<br/>claim_token: "wr_claim_..." }

    Agent-->>Human: Shares claim_token<br/>(prints, MCP skill, etc.)

    Human->>Dashboard: "Claim Agent" button
    Dashboard->>API: POST /v1/agents/claim<br/>(claim_token, humanOnly)
    Note over API: Assigns org_id,<br/>nullifies claim_token
    API-->>Dashboard: { agent: { org_id: "..." } }

    Agent->>API: POST /v1/rooms/:room/check-in<br/>(now claimedAgentOnly passes)
    API-->>Agent: { id: "ci_...", status: "pending" }
```

- **Human-registered** (`POST /v1/agents/register` , humanOnly): Human creates agent, receives API key. Agent belongs to org immediately.
- **Self-registered** (`POST /v1/agents/self-register` , public): Agent registers itself with IP rate limiting. Receives API key + claim_token (`wr_claim_*` , 7-day expiry). Agent has no `org_id` until claimed.

12.4 Data Protection

- Agent API keys stored as salted SHA-256 hashes; raw key shown once at creation, never stored
- Row-Level Security on all 15 tables, org-scoped via `auth_org_id()` helper function
- Audit trail is append-only; no updates or deletes permitted
- Multi-tenant isolation enforced at database level

13. CLI (`wr`)

> ***[NEW SECTION]*** *Not in original spec. The CLI is a first-class consumer of the API, published as* `@waitroom-io/cli` *.*

Binary: `wr` . Built with tsup (ESM, node20). Workspace deps bundled via `noExternal` .

API key resolution: `--api-key` flag → `WAITROOM_API_KEY` env → active profile in `.waitroom/credentials.yaml`

12 command groups: rooms, check-ins, agents, signals, watchers, audit, trust, policies, profile, config, status, whoami.

Output: Auto-detects TTY (table format) vs piped (JSON format).

```
$ wr rooms list
┌──────────────┬────────────────────┬────────────┐
│ ID           │ Name               │ Pending    │
├──────────────┼────────────────────┼────────────┤
│ rm_4x7k2m    │ vendor-payments    │ 3          │
│ rm_8j3n5p    │ code-deploy        │ 0          │
│ rm_2w9q1r    │ default            │ 1          │
└──────────────┴────────────────────┴────────────┘

$ wr check-ins list --room vendor-payments --status pending | jq '.[] | .id'
"ci_8x7k2m"
"ci_3n5p9q"
"ci_1r2w4x"
```

14. Current Capabilities

14.1 Coordination Patterns

Pattern	Status	Description
Agent → Human	✅	Agent creates check-in, human approves/rejects/modifies
Human → Agent (tasks)	✅	Human posts tasks, agents claim/work/submit results with threaded messages
Human → Agent (signals)	✅	Human broadcasts signals, agents receive via watchers
Human → Agent (passive)	✅	Agents watch rooms for events matching trigger conditions
Multi-agent collaboration	✅	Primary + helper agents on tasks via `/join` with weighted trust scoring
Gateway → Sub-agent routing	✅	Gateway manages sub-agents, routes tasks via `agent.task.assigned` WebSocket events
Agent → Agent (delegated)	⚠️	`agent_to_agent` direction and `parent_check_in_id` built; MCP `waitroom_post_subtask` not yet built

14.2 What's Built

Core coordination: - Check-in lifecycle: create → pending → approved/rejected/modified/expired/withdrawn - Bidirectional task flow: human→agent with claim/release/help/join/result lifecycle - Threaded messages on check-ins with typed messages (update/question/answer/result/system) - Multi-agent participation: primary + helper agents with weighted trust scoring - Policy engine: forbid rules → auto-approve rules → trust thresholds → default action - Trust scoring: per-agent-per-room, weights for approval/modification/rejection/expiry - Signals: human broadcasts to rooms, agents subscribe via watchers - Immutable audit trail on every mutation

Agent surfaces: - 27 MCP tools (core 7, task 10, context 2, gateway 8) with Streamable HTTP + OAuth - OpenClaw plugin (13 tools) at `skills/openclaw-plugin/` - TypeScript SDK (`@waitroom-io/sdk-ts`) with `checkInAndWait` , task methods, audit - CLI (`wr` , `@waitroom-io/cli`) — 12 command groups, profile management, TTY-aware output

Human surfaces: - Dashboard (Next.js 15 PWA) — feed, rooms, agent management, Launchpad, audit log, thread drawer - Mobile app (Expo React Native) — check-in feed, push notifications, real-time updates

Sub-agent model: - Gateway agent → sub-agent hierarchy via `gateway_agent_id` FK - `room_agents` table tracking agent-room membership with status - Sub-agent CRUD API + room agent management endpoints (assign, remove, roster) - GatewayConnection class (persistent WebSocket, auto-reconnect, `agent.sync` events) - Agent template gallery (5 templates with full `soul_md`) — deploy in under a minute - Room agent policies: join_mode (auto/approval/invite_only), max_agents, required_capabilities

Infrastructure: - Dual auth: human JWT (Supabase) + agent API keys (salted SHA-256, timing-safe) - Agent self-registration with claim tokens (7-day expiry, IP rate limited) - Real-time: WebSocket, SSE, Redis Pub/Sub, Supabase Realtime (all gracefully degrade) - File uploads: `POST /v1/uploads` + Supabase Storage private bucket (10MB, image/document MIME) - AI task expansion: `POST /v1/tasks/expand` (Claude-powered prompt expansion) - Rate limiting: 60 req/min sliding window per agent (Upstash Redis) - 15 SQL migrations, Row-Level Security on all tables - 5 default rooms seeded per org: General, Code, Research, Email, Files - Skill discovery: `GET /skill.md`, `/skill.json`, `/heartbeat.md`, `/rules.md`

---

15. Roadmap

Item	Description
`waitroom_post_subtask` MCP tool	Agent-to-agent delegation from MCP — post a child task linked to current parent
Task chain guardrails	Enforce max depth and max children per task at API level
Room agent roster UI	Dashboard view of joined/pending/available agents per room
Sub-agent detail page	Agent profile with capabilities, SOUL summary, room assignments, task history
Task tree visualization	Parent/child task chain view in dashboard
Interactive SOUL editing	Rich editing UI for `soul_md` in the dashboard (currently PATCH only)
Trust auto-graduation	

Item	Description
	Wire trust scores into policy engine for automatic approval bypass
Batch approval	Approve/reject multiple check-ins at once from the dashboard
Python SDK	`@waitroom-io/sdk-py`
Escalation chains	Beyond timeout_action: route to backup approvers

16. Open Questions

#	Question	Notes
Q1	Should watchers support semantic/NL triggers?	Current: structured `event_types[]` array only
Q2	Data retention policy for audit trails?	No retention policy or archival logic yet
Q3	How should escalation chains interact with trust?	Escalation chains not yet designed
Q4	Should agents auto-detect sensitive actions?	Currently agent self-classifies risk via check-in
Q5	Rate limiting for agent-posted task chains?	`parent_check_in_id` built but max depth / max children not enforced
Q6	Interactive SOUL editing UX?	API supports PATCH; dashboard needs rich editing UI. Templates as starting point.

17. Success Metrics

17.1 Leading Indicators

- **Time to first loop:** Median time from signup to first completed check-in/approval cycle. Target: < 5 minutes.
- **MCP server installs:** Unique Claude Code / Cursor / Windsurf installations. Target: 500 in first month.
- **CLI installs:** npm installs of `@waitroom-io/cli`. Target: 200 in first month.
- **Daily active rooms:** Rooms with at least one check-in per day. Target: 200 by month 3.
- **Decision latency:** P95 time from human decision to agent notification. Target: < 500ms (WebSocket), < 2s (SSE).

17.2 Lagging Indicators

- **Weekly active agents:** Unique agents with at least one check-in per week. Target: 5,000 by month 6.
- **Modify usage rate:** Percentage of decisions that use Modify (vs. approve/reject). Target: > 15% by month 3.
- **Trust graduation rate:** Agents that graduate to auto-approve for routine actions. Target: 30% by month 6.
- **Retention:** Week 4 retention of orgs that complete onboarding. Target: > 40%.
- **Cross-framework usage:** Orgs using Waitroom across 2+ agent runtimes. Target: > 20% by month 6.
- **Self-registration conversion:** Self-registered agents claimed into an org within 7 days. Target: > 60%.
- **Task routing accuracy:** Tasks where first-matched agent completes without reassignment. Target: > 80%.
- **Agent-to-agent handoff completion:** Child tasks that resolve successfully. Target: > 90%.
- **Multi-agent rooms:** Rooms with 2+ specialized agents. Target: > 30% of active rooms by month 6.

18. Appendices

Appendix	Location
A: Full database schema	15 SQL migrations in `packages/db/migrations/`
B: MCP server tool specs (27 tools)	`apps/mcp-server/src/index.ts`
C: Policy schema	`roomPoliciesSchema` in `packages/shared/src/schemas/`
D: Trust score algorithm	`apps/api/src/services/trust.service.ts`
E: Skill files	Served at `GET /skill.md`, `GET /skill.json`
F: CLI command reference	`apps/cli/src/cli.ts`
G: SDK client methods	`packages/sdk-ts/src/index.ts`
H: Gateway connection	`apps/mcp-server/src/gateway.ts`
I: Upload service	`apps/api/src/services/upload.service.ts`, `apps/api/src/routes/uploads.ts`
J: Agent template library	`apps/dashboard/src/app/(dashboard)/agents/agent-templates.ts`