# HANDS System Management

## The Idea

Traditional dispatch centers rely on manual phone calls, spreadsheet lookups, and calendar checks to assign a technician to a customer. This project replaces that process with an AI agent.

A customer describes their problem in plain language. The AI:

- Understands what kind of service is needed and how urgent it is
- Looks up which employees have the right skills in the database
- Runs a function with an algorithm that checks calendar availability (integrating with Odoo)
- Runs a function that is considered travel time via Google Maps, so travel time to the job site is included in the employee's ticket
- Books the appointment and sends a confirmation email
- Records the ticket in the database

The admin team manages everything through a Vue 3 dashboard — employees, service types, skill levels, areas, and tickets.

All three MCP servers are implemented following Model Context Protocol best practices — each server is a focused, single-responsibility service with clearly typed tool schemas, structured JSON responses, and health endpoints for orchestration readiness.

## What It Can Do

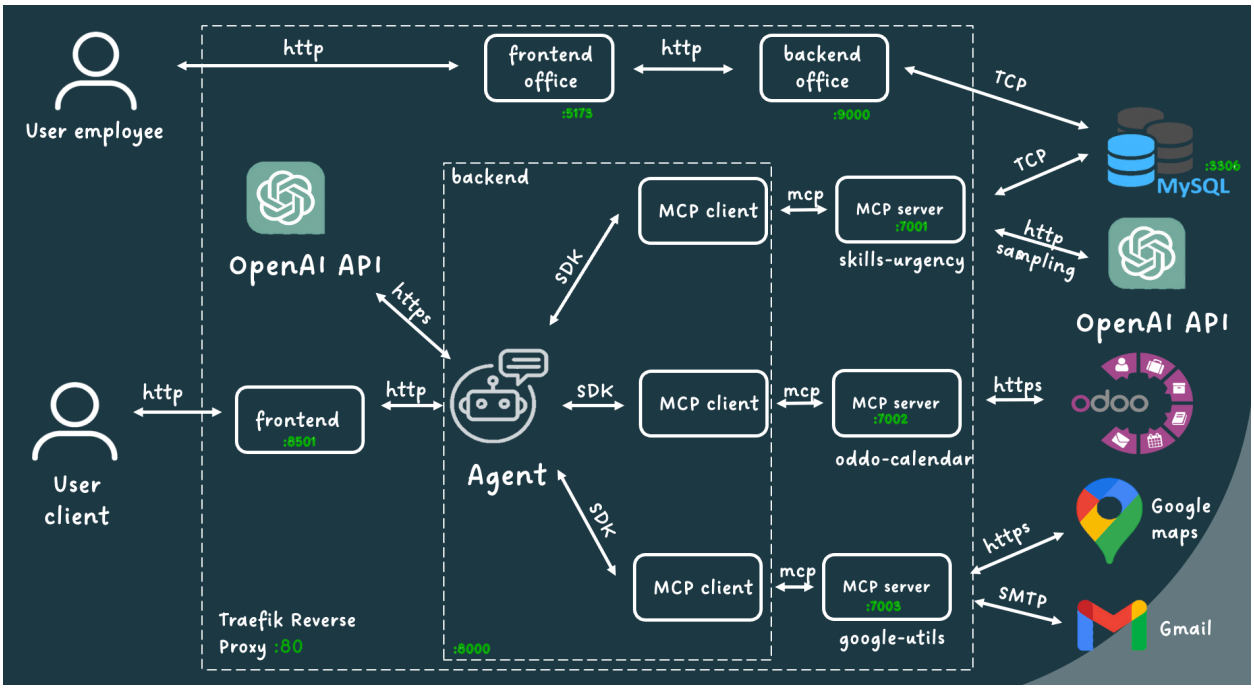
### AI Chat (Customer-facing)

- Accept free-text problem descriptions (e.g. "my kitchen pipe is leaking")
- Classify the problem: service type, affected area, urgency (1–5), required skill level, estimated duration
- Ask clarifying questions only when needed
- Look up all technicians capable of handling the service
- Check real-time calendar availability for multiple employees (Odoo)
- Factor in travel time from company HQ to customer address (Google Maps)
- Present available time slots and let the customer choose
- Create a calendar event for the assigned technician in Odoo
- Send a confirmation email to the customer (SMTP/Gmail)
- Save the ticket to the database with all details
- Update or cancel an existing booking

## Admin Dashboard (Vue 3)

- Login / register with role-based access (ADMIN / USER)
- View and edit own profile
- Browse personal assigned tickets
- Full CRUD for: Employees, Areas, Skill Levels, Service Types, Service Templates, Employee Capabilities, Tickets
- JWT authentication with automatic token refresh (no forced logouts)
- Multi-instance awareness via x-office-instance header

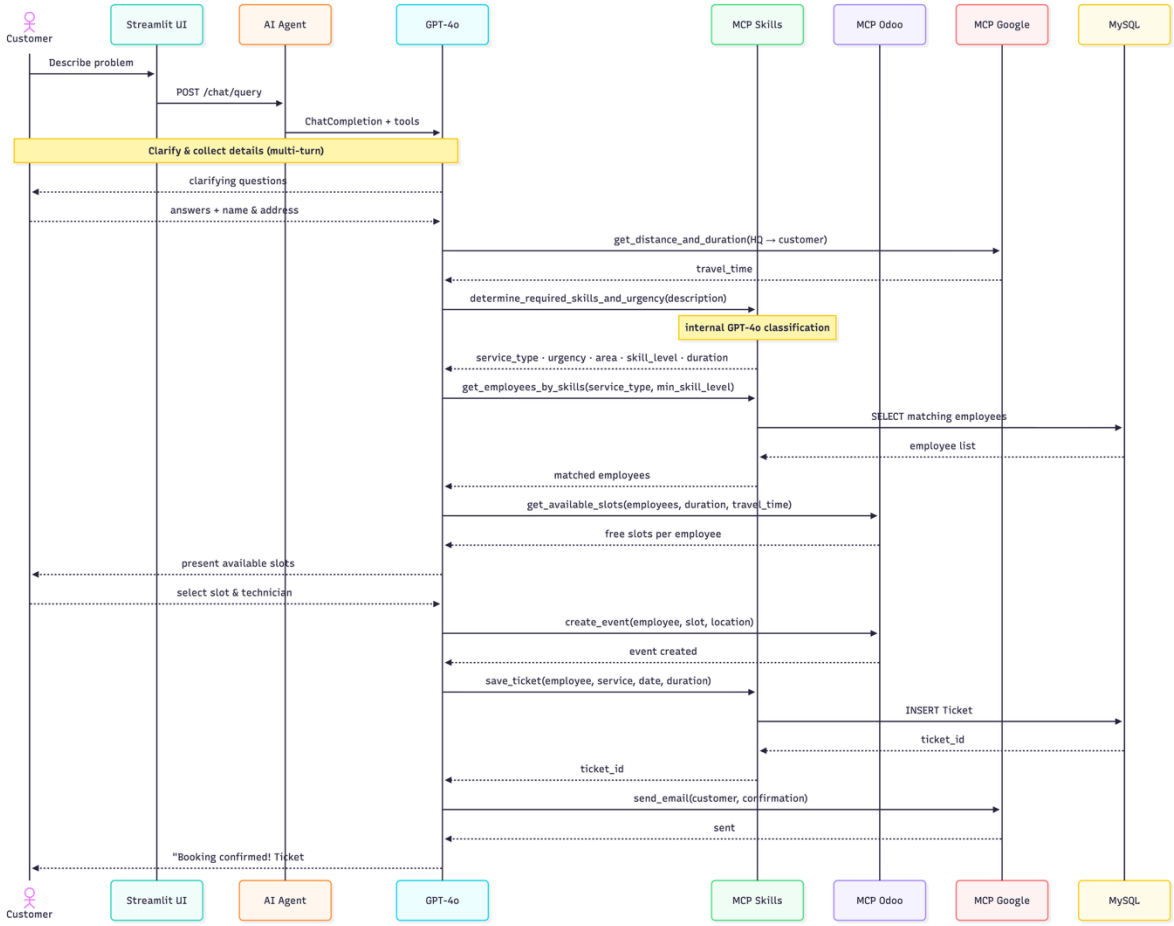
# Architecture Overview



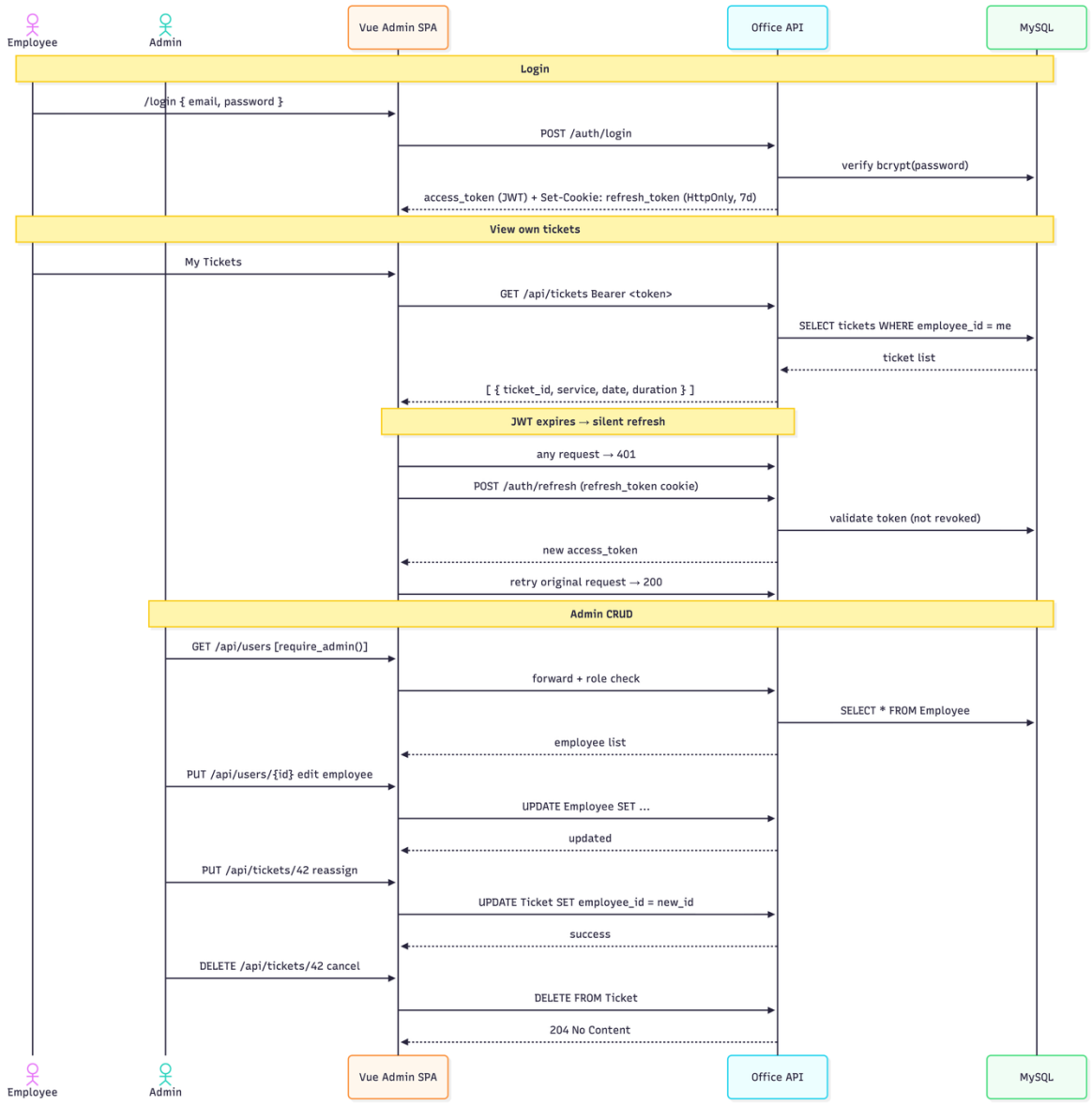

# Sequence Diagram

Full end-to-end interaction covering both the customer chat flow and the employee/admin office flow.

## Part 1 – Customer Chat: AI-Driven Booking

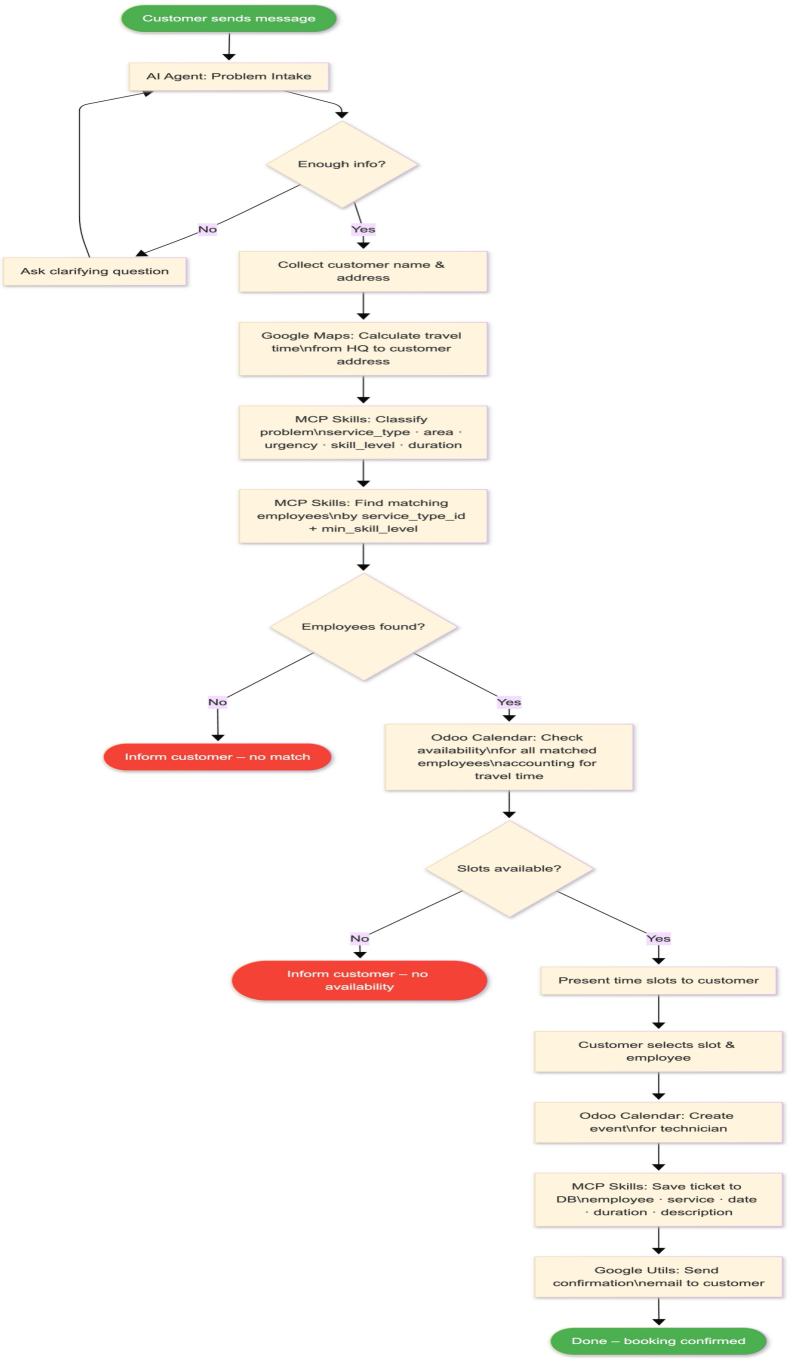



## Part 2 – Office: Employee & Admin Interface Interaction



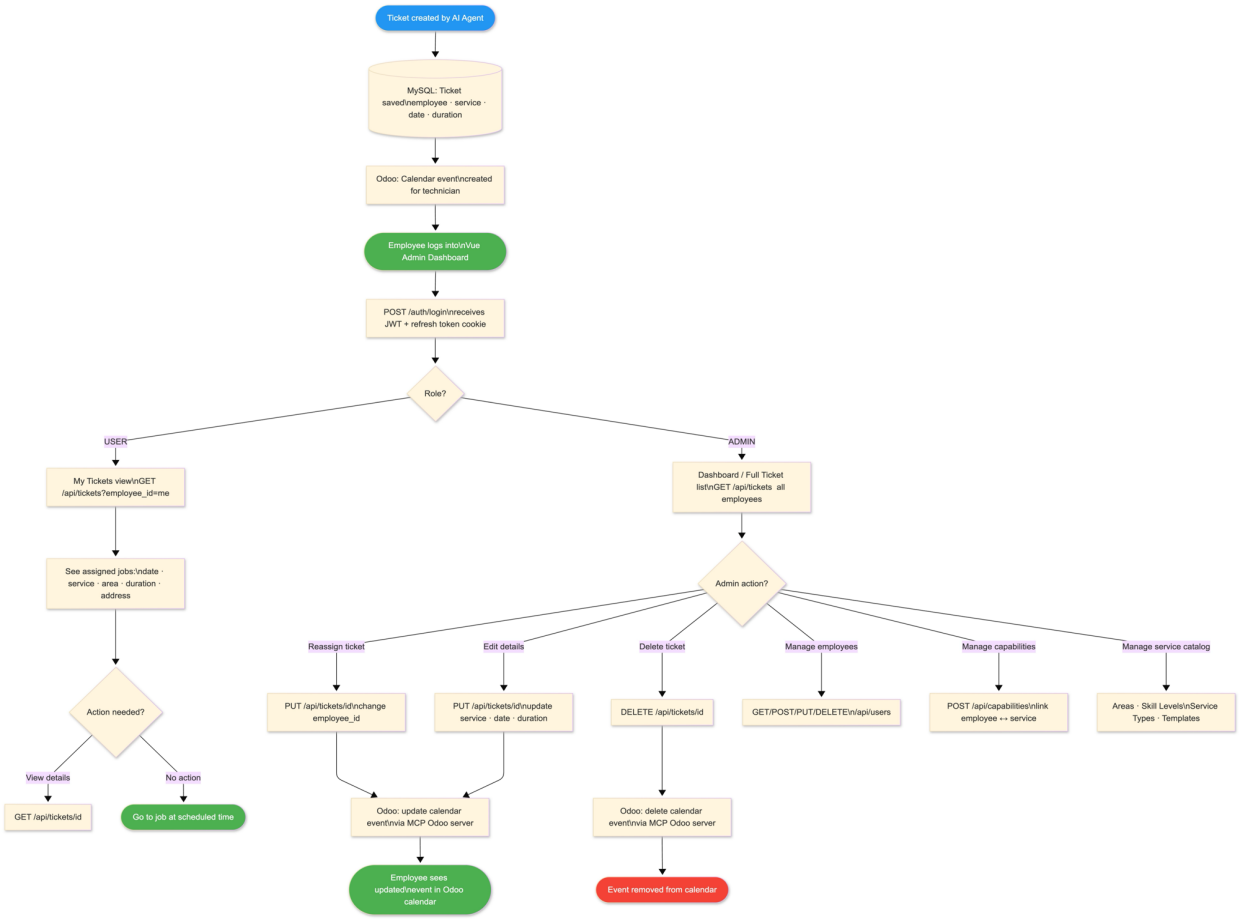

# Workflow Diagram

## Customer Workflow




## Office Workflow — Employee Side

What happens on the employee and admin side after a ticket is created:




## Agent State Machine

The agent follows an explicit state machine defined in agent_instruction.txt:

| State | Description |
|---|---|
| ProblemIntake | Gather problem description from customer |
| ClarifyProblem | Ask follow-up questions if needed |
| CustomerInfo | Collect name and address |
| DistanceCalculation | Google Maps travel time from HQ |
| Classification | GPT-4o classifies service, area, urgency |
| EmployeeSelection | Find technicians matching required skills |
| Scheduling | Check calendar slots for matched employees |
| CalendarEventCreation | Book the appointment in Odoo |
| SendConfirmation | Email the customer |
| TicketCreation | Persist ticket to MySQL |

| | |
|---|---|
| Done / Failed | Terminal states |

# Tech Stack

| Layer | Technology |
|---|---|
| AI Agent | OpenAI GPT-4o via ChatCompletions API |
| Tool Protocol | MCP (Model Context Protocol) / FastMCP 2.x |
| Backend API | FastAPI (Python 3.x) |
| ORM | SQLAlchemy + PyMySQL |
| Auth | PyJWT (HS256) + bcrypt |
| Admin Frontend | Vue 3 + Vite + Pinia + TailwindCSS |
| Chat Frontend | Streamlit (Python) |
| Database | MySQL 8.0 |
| Calendar Integration | Odoo (XML-RPC API) |
| Maps | Google Maps Distance Matrix API |
| Email | SMTP (Gmail App Password or generic) |
| Reverse Proxy | Traefik v3.6.10 |
| Containerization | Docker + Docker Compose |

# MCP Servers (AI Tools)

The AI agent has access to 11 tools across 3 MCP servers. Each server is an independent FastMCP HTTP service.

## Server 1: Skills & Urgency (:7001)

| Tool | Description |
|---|---|
| determine_required_skills_and_urgency | Classifies a problem description using GPT-4o → returns service_name, urgency (1–5), service_type_id, area_id, duration_hours, min_skill_level |
| get_employees_by_skills | Queries DB for employees matching service_type_id and min_skill_level |
| save_ticket | Inserts a new Ticket record into MySQL |
| update_ticket | Updates an existing ticket |

### Server 2: Odoo Calendar (:7002)

| Tool | Description |
|---|---|
| get_available_slots_for_employees_with_time_travel | Finds free calendar slots for a list of employees over the next N days, accounting for travel time |
| create_event_in_oddo_calendar_for_user | Creates a calendar event for a technician in Odoo |
| update_calendar_event_for_user | Reschedules or modifies an existing event |
| delete_calendar_event_for_user | Removes an event from Odoo |

All times use Europe/Zurich timezone.

### Server 3: Google Utilities (:7003)

| Tool | Description |
|---|---|
| get_distance_and_duration | Calls Google Maps Distance Matrix API → returns distance (m/km) and travel duration (s/min) between two addresses |
| send_email | Sends an email via SMTP (Gmail App Password or custom server) |

### MCP Protocol

All servers use JSON-RPC 2.0 over HTTP (FastMCP HTTP transport):

```
POST /mcp/  →  { "method": "tools/call", "params": { "name": "...",
"arguments": {...} } }
GET  /health  →  { "status": "ok" }
```

## Frontend Applications

### Vue 3 Admin Dashboard (/)

A full SPA for administrators and employees.

#### Public Routes

- /login — JWT login form
- /register — Self-registration

#### Authenticated Routes (any role)

- /profile — View & edit own profile
- /dashboard — Overview
- /my-tickets — Tickets assigned to the current user

### Admin-only Routes

- /employees — Employee list, create, edit, delete
- /areas — Area management
- /skill-levels — Skill level management
- /service-types — Service type management
- /service-templates — Service order templates
- /capabilities — Employee–service capability mappings
- /tickets — All tickets

## Streamlit Chat UI (/streamlit)

A conversational interface where the customer describes their problem and interacts with the AI agent step-by-step. Sends messages to POST /chat/query and displays streamed responses.

# Authentication & Security

## JWT Token Lifecycle

| Step | Description |
|---|---|
| 1. Login | POST /auth/login |
| 2. Token issued | Access token (HS256) + Refresh token (7 days, HttpOnly cookie) |
| 3. Storage | Access token stored in localStorage ('hands_auth_token') |
| 4. Requests | Authorization: Bearer <token> attached to every /api/* request |
| 5. On 401 | Client calls POST /auth/refresh → retry request on success |
| 6. App reload | GET /auth/verify validates stored token |

## Security Notes

- Passwords hashed with bcrypt (12 rounds) — never stored in plaintext
- JWT signed with configurable secret (minimum 32 chars enforced by Pydantic)
- Stack traces never exposed in API error responses (global exception handler)
- All .local.env files are gitignored — secrets never committed
- CORS currently allows all origins (restrict in production)

# API Reference

## Base URLs

| Environment | Base URL |
|---|---|
| Local dev | http://localhost:9000 |

| | |
|---|---|
| Docker | http://localhost (via Traefik) |

## Authentication Header

```
Authorization: Bearer <access_token>
```

## Endpoints

### Auth

```
POST   /auth/register        Public
POST   /auth/login           Public — returns access token + sets refresh
cookie
POST   /auth/refresh         Cookie — returns new access token
POST   /auth/logout          Cookie — revokes refresh token
GET    /auth/verify          Bearer — validate token (used by Traefik)
GET    /auth/health          Public
```

### Employees

```
GET    /api/users            ADMIN  ?page=1&size=20
GET    /api/users/{id}       ADMIN
POST   /api/users            ADMIN
PUT    /api/users/{id}       ADMIN
DELETE /api/users/{id}       ADMIN
```

### Domain Entities (all ADMIN)

```
GET/POST/PUT/DELETE  /api/areas/{id}
GET/POST/PUT/DELETE  /api/skill-levels/{id}
GET/POST/PUT/DELETE  /api/service-types/{id}
GET/POST/PUT/DELETE  /api/service-templates/{id}
GET/POST/PUT/DELETE  /api/capabilities/{id}
GET/POST/PUT/DELETE  /api/tickets/{id}
```

### AI Agent

```
POST   /chat/query           { "message": "...", "history": [...] }
GET    /chat/tools           Lists all available MCP tools
```

# Database Schema

| Table | Key Fields |
|---|---|
| Employee | employee_id, abbrev, email, password_hash, name, surname, phone, gender, address, salary, role (ADMIN\|USER), skill_level_id, service_type_id |
| ServiceType | service_type_id, name (e.g. Plumbing, Heating, Ventilation) |
| Area | area_id, name (e.g. Kitchen, Bathroom, Basement) |
| SkillLevel | skill_level_id, level (1–5) |

| | |
|---|---|
| ServiceOrderTemplate | service_id, area_id, service_type_id, service_name, description, duration_hours, min_skill_level |
| EmployeeServiceCapability | capability_id, employee_id, service_id (Many-to-many: employee ↔ service) |
| Ticket | ticket_id, employee_id, service_id, date, duration_hours, description, department, created_at |
| RefreshToken | token_id, employee_id, token_hash, expires_at, revoked |

# Installation & Configuration

## Prerequisites

- Python 3.11+
- Node.js 18+ (for Vue frontend)
- MySQL 8.0 (local dev) or Docker (recommended)
- OpenAI API key
- Odoo instance with API access
- Google Maps API key
- Gmail App Password (or SMTP credentials)

## Environment Files

Copy each .env template to .local.env and fill in the values.

### Root .local.env

```
MYSQL_PASSWORD=
MYSQL_ROOT_PASSWORD=
OPEN_AI_SECRET_KEY=
```

### backend/.local.env

```
INSTRUCTION_PATH=agent_instruction.txt
OPEN_AI_MODEL=gpt-4o
OPEN_AI_SECRET_KEY=
OPENAI_BASE_URL=https://api.openai.com/v1
TRANSPORT=http
BASE_URL_MCP_SKILLS_URGENCY_SERVER=http://127.0.0.1:7001
BASE_URL_MCP_ODDO_CALENDAR_SERVER=http://127.0.0.1:7002
BASE_URL_MCP_GOOGLE_UTILS_SERVER=http://127.0.0.1:7003
JWT_SECRET=<min 32 chars>
JWT_ALGORITHM=HS256
JWT_EXPIRE_SECONDS=3600
MYSQL_HOST=127.0.0.1
MYSQL_USER=root
MYSQL_DB=hands_system
MYSQL_PORT=3306
```

# Running the Project

## Option A: Docker (Recommended)

```
docker compose -p hands-system up -d --build
```

URLs after startup:

| Service | URL |
|---|---|
| Vue Admin Dashboard | http://localhost |
| Streamlit Chat | http://localhost/streamlit |
| Office API | http://localhost/api |
| Auth API | http://localhost/auth |
| AI Agent | http://localhost/chat |
| Traefik Dashboard | http://localhost:8888 |

## Option B: Local Development

### 1. Set up Python environment

```
python3 -m venv .venv
source .venv/bin/activate
pip install -r requirements.txt
```

### 2. Start MySQL

```
brew services start mysql
```

### 3. Initialize the database

```
mysql -u root -p hands_system < database/schema.sql
mysql -u root -p hands_system < database/fill_hand_system.sql
```

### 4. Start MCP servers (3 separate terminals)

```
python -m services.mcp_skills_urgency.server    # :7001
python -m services.mcp_oddo_calendar.server     # :7002
python -m services.mcp_google_utils.server      # :7003
```

### 5. Start the AI agent backend

```
cd backend
uvicorn src.agent.agent:app --port 8000 --reload
```

### 6. Start the Office API

```
cd backend/office
uvicorn app:app --port 9000 --reload
```

### 7. Start the frontends

```
cd frontend-vue && npm install && npm run dev     # http://localhost:5173
streamlit run frontend/src/main.py --server.port=8501
```

# Makefile Commands

| Command | Description |
|---|---|
| make build | Build all Docker images |
| make up | Build + start entire stack (office-api scaled to 2 replicas) |
| make stop | Stop all containers — volumes and data preserved |
| make down | Stop and remove containers — volumes preserved |
| make clean | Full reset — removes containers, networks, and volumes |
| make logs | Tail logs for all key services |
| make logs-service SERVICE=<name> | Tail logs for a specific service |
| make restart SERVICE=<name> | Restart one service without rebuilding |
| make test | Run pytest inside the office-api container |
| make loadtest | k6 load test — ramps 5 → 10 → 20 VUs (~90s) |
| make loadtest-stress | k6 stress test — ramps up to 50 VUs to find breaking point |
| make help | Print all available commands with descriptions |

# Docker Services & Routing

| Route | Service | Priority |
|---|---|---|
| /api* /auth* | Office API :9000 × 2 replicas | 30 |
| /chat* | AI Agent :8000 | 20 |
| /streamlit* | Streamlit :8501 | 20 |
| / | Vue SPA :8080 | 1 |

## Service Health Checks

- db-migrate depends on mysql-db (healthy)
- backend-chat depends on all 3 MCP servers (healthy)
- office-api depends on db-migrate (completed)
- frontends depend on office-api (healthy)

## Important Notes

- The MYSQL_HOST value must be mysql-db in Docker and 127.0.0.1 for local dev
- All .local.env files are gitignored — never commit secrets
- MCP base URLs in backend/.local.env must point to 127.0.0.1:700x for local dev and Docker service names when running in Docker — docker-compose.yaml overrides these automatically
- The agent's behavior is fully defined in backend/src/agent/agent_instruction.txt — edit this file to change how the AI agent handles conversations
- JWT expiry is configurable via JWT_EXPIRE_SECONDS in backend/.local.env

## References


- FastMCP Documentation: https://gofastmcp.com/integrations/fastapi
- Model Context Protocol Best Practices: https://modelcontextprotocol.info/docs/best-practices/
- OpenAI Function Calling: https://platform.openai.com/docs/guides/function-calling
- Odoo External API: https://www.odoo.com/documentation/17.0/developer/reference/external_api.html