HOA Chat System — Comprehensive Technical Checkpoint

1. Full System Stack Overview

Cost Summary: ≈ **$5/month base (VPS)** + **OpenAI API usage**

---

2. Backend Architecture — Modules & Responsibilities

app.py

Entrypoint (FastAPI / Flask). Defines routes:
- `/login/start`, `/login/verify`, `/session`, `/ask`
- Delegates to `auth`, `qa`, and `validator` modules.

auth.py

Handles user authentication and session persistence.
- `start_login(email)` → send magic link
- `verify_login(token)` → create session + cookie
- `get_tier_from_request(request)` → resolve user tier (PUBLIC / OWNER / BOARD)

roster.py

Access control registry (AccessRoster + Sessions tables).
- `get_tier_for_email(email)` → lookup tier
- `is_valid_session(session_id)` → verify session

policy_engine.py

Builds OpenAI Responses API call per tier.
- Selects vector stores and web search tools.
- Embeds strict instruction block:
- 3■line answer format
- Legal hierarchy enforcement
- “Most restrictive lawful rule controls.”
- Tool selection logic:
- PUBLIC → public stores + law tools
- OWNER → + private_static + private_dynamic
- BOARD → + privileged_dynamic

openai_client.py

Low-level API client.
- `run_qa(oai_request)` → calls OpenAI Responses API
- Returns `draft_answer`, `tool_trace`

validator.py

Post■processing gatekeeper.
- Checks format (3■line structure)
- Verifies explicit hierarchy phrasing (“Oakland law controls…”)
- Ensures citations cover every tool used
- Confirms source order: federal → state → county → city → CC&Rs; → HOA rules
- Enforces tier leak prevention
- Returns safe fallback on violation

qa.py

Pipeline orchestrator.
- `answer_question(question, tier)`
1. Build OpenAI request (policy_engine)
2. Execute (openai_client)
3. Validate (validator)
4. Log (audit)
5. Return answer

audit.py

- `log_interaction(...)` → timestamp, email, tier, question, answer, tool_trace, validator result

emailer.py

- `send_magic_link(email, token)` → via SMTP provider

config.py

Holds constants:
- Vector store IDs
- Domain whitelists
- Cookie TTL
- OpenAI key
- Hierarchy order
- Fallback text templates

---

3. Tools Call Definitions

3.1 Vector Store Set (5 total)

3.2 Web Search Groups (4 total)

Hierarchy of authority enforced:
Federal → State → County → City → CC&Rs; → HOA Rules/Policies → Board/Privileged Docs

---

4. Processing Flow

1. User → Google Sites iframe → `/ask` API.
2. Backend reads cookie → resolves tier via AccessRoster.
3. Policy engine builds allowed tool list + strict instructions.
4. OpenAI Responses API runs (5 vector stores, 4 web■search groups).
5. Validator checks hierarchy, formatting, access, citations.
6. Audit log stores trace and final answer.
7. Answer returned to iframe.

---

5. Logical Diagram (Text)

```
[ Google Sites ]
↓ (iframe)
[ Netlify Chat UI ]
↓ HTTPS JSON
[ Python Backend (VPS) ]
■■ Auth (magic links, cookies)
■■ AccessRoster (Supabase)
■■ PolicyEngine → OpenAI
■■ Validator (hierarchy & leak guard)
■■ AuditLog → Supabase
↓
[ OpenAI Cloud ]
■■ VectorStores (5)
■■ WebSearch Tools (4)
■■ GPT■5 Responses → validated answer
```

---

6. Tier Matrix Summary

---

7. Cost Structure

Typical total: ≈ **$5 + API usage per month**

---

8. Future / Optional Add■Ons

- **Owner Portal UI:** lightweight dashboard for BOD to upload new policy PDFs (auto■vectorized).
- **Alert System:** when CC&R; updates detected → retrain vector store.
- **Periodic Validation:** re■run test queries weekly; flag mismatched hierarchy outputs.
- **Version Tags:** attach effective date metadata to dynamic docs.

End of Checkpoint