Documenting

every live endpoint (with its inputs, outputs, auth surface, and service‐level responsibilities) gives us a concrete map we can use to

prioritize the refactoring phases.

Below is a per-service cheat-sheet of all the HTTP endpoints you have in place today, their purpose, how to call them, and any special

notes. I’ve also included ports and key environment variables so a new developer can spin up the whole stack end-to-end.

–––

SERVICE 1: Auth Service

• Location: microservices/auth\_service

• Default port: 8014

• Env vars: none (just spins up)

• Endpoints:

– GET /health

• Returns

{ "status": "ok", "service": "auth" }

• Use: simple liveness/readiness check for deployments.

SERVICE 2: DB Service

• Location: microservices/db\_service

• Default port: 8013

• Env vars (in .env or environment):

POSTGRES\_DB, POSTGRES\_USER, POSTGRES\_PASSWORD, POSTGRES\_HOST, POSTGRES\_PORT, DB\_CONNECT\_MAX\_RETRIES, DB\_CONNECT\_RETRY\_INTERVAL

• Endpoints:

– GET /health

• Returns

{ "status": "ok", "service": "db" }

• Use: confirm connectivity to Postgres (actual CRUD lives in code but isn’t exposed yet—you’ll want to add /users, /sessions,

etc.)

SERVICE 3: RAG Service

• Location: microservices/rag\_service

• Default port: 8010

• Env vars: none beyond standard Chroma path (‘chroma\_db’ folder)

• Endpoints:

– GET /health

• Returns

{ "status": "ok", "service": "rag" }

– GET /query/query

• Query parameters:

• prompt (str, required)

• top\_k (int, default=3)

• tags (csv list, optional)

• Returns

{ "result": "<concatenated text from top-K docs>" }

• Use: backwards-compatible RAG search.

– GET /rag/query

• Same signature & behavior as /query/query (just a second mount point).

• (NOTE: there is a “api.py” in this folder that defines a richer chat-orchestration layer—GET /query?… and POST /generate—which is

currently not wired into main.py. We should decide ASAP whether to bring those routes live here or move them to a dedicated chat

service.)

SERVICE 4: Storage (“Session”) Service

• Location: microservices/storage\_service

• Default port: 8011

• Env vars: none (all file-based/Pydantic models)

• Routers included under:

prefix=“/aq” ⇒ AQ scoring

prefix=“/session” ⇒ reflection

prefix=“/profile” ⇒ user-profile CRUD

• Endpoints:

– GET /health

• Returns service status (note: currently reports "service": "rag"—we should fix to "storage")

– POST /aq/api/session/aq

• Body (JSON):

{

user\_id: str,

reflection\_depth: int,

action\_responsiveness: int,

recalibration\_speed: int,

alignment\_literacy: int,

self\_initiated\_behavior: int,

feedback\_integration: int,

emotional\_regulation: int

}

• Returns

{ aq: float, tier: str, message: str }

• Use: record and classify Alignment Quotient.

– POST /session/api/session/reflection

• Body (JSON): { user\_id: str, reflection: str }

• Returns whatever handle\_reflection(...) returns.

• Use: store and process end-of-turn reflections.

– GET /profile/api/user/{user\_id}

• Path param: user\_id (str UUID)

• Returns full UserProfile JSON (Pydantic model).

– PUT /profile/api/user/{user\_id}

• Body (JSON): { name?: str }

• Returns updated UserProfile JSON.

SERVICE 5: Stats Service

• Location: microservices/stats\_service

• Default port: 8012

• Env vars: none

• Endpoints:

– GET /health

• Returns { "status": "ok", "service": "stats" }

– GET /stats/test

• Returns { "message": "Stats route is alive" }

• Use: basic sanity check for analytics ingestion.

SERVICE 6: Shared Library

• Location: microservices/shared

• Not itself a server—packages out common cache, metrics, tag inference, prompt\_engine, agents, etc.

• To consume: add PYTHONPATH=/app or install as a pip editable.

–––

Authentication Notes

• The only routes today that enforce an API key are in the chat-orchestration layer in api.py (X-XAVIGATE-KEY header). All other

services are wide open.

• For MVP you’ll want to:

– Wire auth\_service into each service (e.g. via HTTP call or FastAPI dependency).

– Centralize key storage in env var XAVIGATE\_KEY (or switch to JWTs once you have user login).

MVP-Critical Endpoints (suggested priority)

1. Chat /generate and/or GET /query?… with auth header

2. RAG search (GET /query/query)

3. Profile load and update (GET/PUT /profile/api/user/{id})

4. Session logging (POST /session/api/session/reflection)

5. AQ scoring (POST /aq/api/session/aq)

6. Health checks & stats (for deployment pipelines)

Next Steps

• Please confirm that these match what you see in your running containers, and whether you’d like to fold in the richer

chat-orchestration endpoints from api.py now or postpone that until after the six-service split.

• Let me know if any endpoints are missing, or if you’d like example curl invocations, OpenAPI snippets, or a Postman/Insomnia

collection generated.