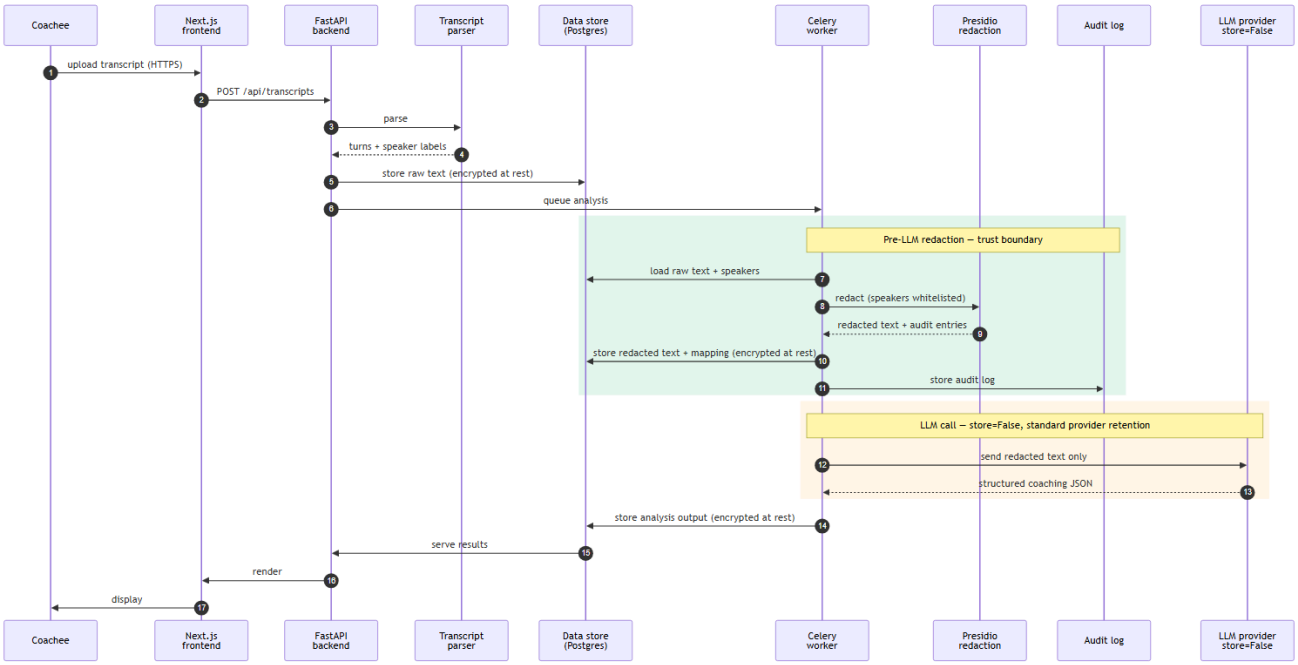
MAKING YOURSELF **CLEAR**

DATA HANDLING

Making Yourself Clear Data Flow

This document describes how transcript data flows through the Making Yourself Clear system, from upload through analysis to storage. Technical claims are sourced from the code; see file references for verification. Last reviewed: **2026-07-08** (refreshed for at-rest encryption, AWS KMS rooting, and the current OpenAI retention posture).

Diagram



Steps 7–11 (highlighted in teal) sit inside the trust boundary: this is where redaction is applied before any data leaves the worker. Steps 12–13 (highlighted in amber) cross the trust boundary to the LLM provider; every request sets `store=False` (which disables OpenAI-side conversation storage — see *LLM call* below for the full retention posture). **When redaction is enabled (the default), no transcript text reaches the LLM provider without passing through the redaction layer.** All content written back to the data store (steps 5, 10, 14) is encrypted at rest (application-layer AES-256-GCM under a per-organization key).

Step-by-step

Upload and storage (steps 1–5)

A coachee uploads a transcript file (txt, vtt, srt, docx, pdf, csv, json, md) from the frontend. The backend parses the file and stores the raw text in the data store, indexed against the coachee's account. The text is **encrypted at rest on write** (application-layer AES-256-GCM under a per-organization key; see *Locations of processing* / the white paper §6.2). **The original transcript text is not sent to any third-party service at this stage.**

- Endpoint: `backend/api/routes_transcripts.py`
- Storage: `transcripts.raw_transcript_text` column in the Postgres database, managed via `backend/core/postgres_datastore.py`; stored as an `enc:v1:` AES-256-GCM envelope under the org's data-encryption key (`backend/core/org_data_keys.py`).

Redaction (steps 6–11)

When the coachee requests an analysis, a Celery worker loads the raw text and the list of speaker labels extracted from the transcript. The redaction layer (`backend/core/pii_redactor.py`) uses Microsoft Presidio with a set of built-in and custom recognizers to detect personally identifiable and sensitive information.

What is redacted:

Category	Examples
Names of people **not in the meeting**	Anyone mentioned but not a speaker (customers, competitors, family).
Email addresses	Standard email-address patterns.
Phone numbers	US and international formats.
Physical addresses	US / Canada / UK street-address patterns.
Government IDs	SSN, passport, driver's licence, medical licence.
Credit card numbers	Full numbers and partial references (e.g. "Amex ending in 1234", expiration dates).
Employee IDs	EMP-####, EMPID-####, STAFF-####, etc.
Dates of birth	Explicit DOB references.
API keys / credentials	OpenAI keys, Stripe keys, webhook secrets, labelled secrets.
IP addresses	IPv4, IPv6.
URLs	Web addresses.
Locations	Cities, regions, countries (Conservative aggressiveness only).
Nationalities / religions / political groups	Detected as NRP entities.

What is intentionally not redacted:

- **Names of meeting participants (speakers).** Each transcript's speaker labels are added to a do-not-redact allowlist before Presidio runs (`backend/core/workers.py:549`, `backend/core/pii_redactor.py:374-379`). The full name plus individual name tokens are protected. Reason: the LLM needs to refer to speakers by name to produce coherent coaching feedback (e.g. "Sarah, you interrupted Daniel twice in the first five minutes"). The privacy gain from redacting speakers is small (the coachee's own name is already in their Making Yourself Clear account), while the coaching-quality loss would be material.
- **Organization names.** Off by default; can be enabled per-deployment via the admin redaction settings.
- **Structural metadata** (timestamps, turn numbering, speaker role labels). Does not identify individuals.

Aggressiveness levels: Conservative, Standard (default), Permissive. Conservative detects the most (including generic locations); Permissive limits redaction to high-risk identifiers (SSN, credit cards, credentials). Configurable globally by the vendor admin and overridable per-organization by an org admin (five settings: enabled, aggressiveness, reversible-mapping, redact-organization-names, share-original-with-coach).

What is stored after redaction:

- The redacted text — the exact bytes sent to the LLM.
- A reversible token-to-original mapping (e.g. `<PERSON_1>` → `"Mark Smith"`). Stored to support audit and to allow operators to verify what was redacted on any given run. **Not used to substitute original values back into the AI's response.**
- A per-entity audit log capturing entity type, token, turn ID, and character positions.

LLM call (steps 12–13)

The redacted text — and only the redacted text — is sent to the configured LLM provider. The provider is selected by Making Yourself Clear's LLM router (`backend/core/llm_client.py`) based on the configured model name: models starting with `claude-` route to Anthropic, all others route to OpenAI. Deployments can configure a custom endpoint (Azure OpenAI, OpenAI-compatible servers, etc.) via the `OPENAI_BASE_URL` / `ANTHROPIC_BASE_URL` environment variables at the vendor level, OR via the per-organization BYO LLM settings (preferred for multi-tenant customers).

Provider retention controls:

- **OpenAI (default):** `store=False` is set on every request (`backend/core/openai_client.py`), which disables OpenAI-side storage of the conversation. OpenAI does not train on API data (standard API terms); standard API retention of up to 30 days (abuse-monitoring) applies. A Zero-Data-Retention agreement has been **applied for and is pending** — it is not yet in effect, so we do not claim zero retention today.
- **Anthropic (Claude models only, not the default):** Anthropic's standard commercial API policy is no training on API data and up-to-30-day operational retention. Customers wanting stronger controls should use BYO LLM with their own enterprise agreement.
- **BYO LLM endpoints (per-organization):** An org_admin can configure their own provider (`openai` or `anthropic`), endpoint URL, model, and API key in the in-app Settings page. The API key is encrypted at rest with AES-256-GCM under the deployment root key `APP_ENCRYPTION_KEY` (which on SaaS is held in AWS KMS). At LLM-call time, the worker resolves the coachee's organization, decrypts the API key momentarily, and routes the call through the customer-controlled endpoint. Retention is then governed by the customer's contract with their LLM provider; only the structured response is stored back in Making Yourself Clear's database (step 14).

Storage and display (steps 14–17)

The structured coaching JSON is stored against the run record. The frontend fetches and renders it for the coachee. Coaches assigned to the coachee can view the same results.

Data subject access

- **Coachees** can view, download, and delete any of their own transcripts and runs via the run detail page.
- **Coaches** can view (but not delete) transcripts and runs belonging to coachees assigned to them.
- **Org admins** can view all data belonging to coachees in their organization, manage members, and configure per-org privacy settings (PII redaction overrides, BYO LLM endpoint).
- **Vendor admins** (Making Yourself Clear operators) can view all data across all organizations through the admin UI. Admin actions (role changes, redaction configuration changes, BYO LLM changes, member invites/removals) are recorded to an append-only `admin_audit_log` table in the application database, capturing actor identity, action, target, timestamp, and originating IP / User-Agent. The BYO LLM API key value itself is never recorded — only the action ("set" / "rotated" / "cleared"). Read-only access to user data (browsing the admin UI without taking an action) is not currently audited at the application layer; this is on the SOC 2 roadmap.

Locations of processing

- **Application servers (backend, worker, frontend):** Railway, US West region (California). Railway runs on Google Cloud Platform infrastructure.
- **Application database (Postgres):** Railway managed Postgres, co-located in the same US West region. Holds all customer data (transcripts, runs, baseline packs, experiments, audit log, organizations, system configs), with customer content **encrypted at rest** (application-layer AES-256-GCM under a per-organization data-encryption key).
- **Key management (AWS KMS):** US East (Ohio, us-east-2). Holds the deployment root key that wraps the per-org data-encryption keys and at-rest secrets; every unwrap is a KMS Decrypt (audited, revocable). Content bytes never leave the app for KMS — only key material is wrapped/unwrapped.
- **LLM provider (OpenAI):** US, OpenAI's default routing. Requests sent with `store=False`.
- **LLM provider (Anthropic):** US, when a Claude model is selected via the model-prefix router or via a per-org BYO LLM configuration.
- **Centralised logs:** Better Stack (EU data region on the current plan).
- **Error tracking:** Sentry (`*.sentry.io`, US ingest).
- **Transactional email:** SendGrid.
- **Source maps / build artefacts:** Sentry (US ingest) for sourcemap upload; GitHub for source control.

Verification

The technical claims above can be re-verified by inspecting:

- `backend/core/workers.py`: redaction invocation and storage of redacted artefacts.
- `backend/core/postgres_datastore.py` + `backend/core/org_data_keys.py`: application-layer at-rest encryption (per-org AES-256-GCM DEK) at the DataStore write chokepoint.
- `backend/core/pii_redactor.py`: Presidio configuration, custom recognizers, allowlist mechanism.
- `backend/core/openai_client.py` / `backend/core/anthropic_client.py`: LLM client construction including retention parameters.
- `backend/api/routes_runs.py` (`get_run_redaction`): the endpoint surfacing the redacted text and entity counts to the in-app "View what was sent to the LLM" panel.