# RAI Assessment Copilot – Solution Description

This document summarizes the current implementation of the RAI Assessment Copilot and highlights the key controls for security, data privacy, and Responsible AI (RAI) risk remediation. It is intended to act as the baseline for compliance review against the [Microsoft Responsible AI Standard – General Requirements](https://cdn-dynmedia-1.microsoft.com/is/content/microsoftcorp/microsoft/final/en-us/microsoft-brand/documents/Microsoft-Responsible-AI-Standard-General-Requirements.pdf?culture=en-gb&country=gb) and to record residual risks and planned improvements.

## 1. Purpose and Architecture

* **Goal**: accelerate creation of Microsoft Responsible AI (RAI) Impact Assessment drafts by guiding trained assessment owners through a multi-step, model-assisted workflow. The Copilot produces both Microsoft-internal and public-facing DOCX outputs that follow the official RAI template.
* **Primary users**: Microsoft RAI champions and solution owners who have completed the mandated RAI training. Access is limited to this population through Entra ID authentication and a Key Vault–backed allow-list.
* **User flows**: - **Streamlit UI (streamlit\_ui\_main.py)** – interactive experience for uploading a solution description, running an AI-assisted pre-flight analysis, generating draft assessments, and downloading outputs or log bundles. - **HTMX/FastAPI UI (htmx\_ui\_main.py)** – web application with CSRF protection, live progress polling, toast notifications, and a remediation dashboard that gates uploads on malware scans, Azure Content Safety verdicts, prompt sanitization, and Azure AI Language PII findings before allowing generation. - **CLI (main.py)** – headless generation path for scripted scenarios.
* **Pipeline orchestration**: prompts/prompts\_engineering\_llmlingua.py executes a deterministic 12-step prompt pipeline (Intended Uses → Disclosure of AI Interaction). Each step transforms the template via typed JSON results, with fallbacks to text when needed.
* **Key integrations**: Azure OpenAI (Responses API via the official AzureOpenAI client), Azure Content Safety Prompt Shields, Azure AI Language (PII entity detection), Azure Key Vault (secrets, user/administrator allow-lists), Azure Blob Storage (access logs and admin downloads), optional llmlingua prompt compression, ClamAV malware scanning, and local filesystem outputs (rai-assessment-output/).
* **Deployment targets**: designed for Azure Container Apps with Managed Identity; supports local execution (Mac/Linux/WSL2) for development and air-gapped testing when Key Vault access is extended through the network security perimeter.

## 2. Security Controls

### 2.1 Identity and Access Management

* **Interactive sign-in**: MSAL custom Streamlit component enforces Entra ID authentication. The login surface clearly links to the hosting Azure Container App URL.
* **Authorization**: a Key Vault secret (RAI-ASSESSMENT-USERS) contains the approved user allow-list. Users outside the list are denied access to generation features.
* **Admin privileges**: elevated actions (model selection, reasoning verbosity, log level changes, cache clearing) are only available to named administrators validated against the allow-list.
* **Session integrity**: HTMX routes require a per-session CSRF token injected into forms and headers; POST requests missing or mismatching the token are rejected, limiting CSRF exploitation.

### 2.2 Secrets and Key Management

* **Keyless pattern**: DefaultAzureCredential retrieves tokens for Azure OpenAI, Blob Storage, and Key Vault. No API keys are stored in the repository or configuration files.
* **Secret storage**: connection details (OpenAI endpoint/deployment, Blob account, allow-list) are sourced from Key Vault or container environment variables injected at deploy time.
* **Unique output paths**: every run receives an MD5-based identifier to prevent cross-session data leakage in the shared rai-assessment-output/ directory.

### 2.3 Network and Platform Security

* **Azure Container Apps**: recommended hosting pattern with Managed Identity and network security perimeter (NSP) support for private Key Vault access. Local execution scenarios require the NSP to approve the developer’s public IP or provide VPN access to the private endpoint (see runbook in repository documentation).
* **Dependency hygiene**: Python dependencies pinned in requirements.txt; container images built via provided Dockerfile with hardened base image expectations (CVE scanning performed in CI/CD pipeline before production release – TODO to automate).
* **Process isolation**: Uploaded documents are parsed inside a resource-constrained worker process (CPU, memory, and wall-clock limits) so hostile DOCX/PDF payloads cannot exhaust the main process.

### 2.4 Logging, Monitoring, and Auditability

* **Structured logging**: helpers/logging\_setup.py configures rotating file logs (./logs/app.log, DEBUG level) plus console output with adjustable verbosity. Noisy third-party libraries are suppressed to WARNING.
* **Action logging**: helpers/blob\_cache.append\_log\_to\_blob appends timestamped user actions (login, parameter changes, downloads) to an Azure Blob. Administrators can export these logs or download packaged system logs (ZIP) from the UI for audit investigations.
* **Reasoning transparency**: administrators can opt-in to display reasoning summaries returned by the Responses API, supporting post-hoc review.
* **Cost telemetry**: each run displays the calculated token cost (prompt, completion, reasoning tokens) to reinforce responsible usage and budgeting.

### 2.5 Threat Scanning & Remediation Pipeline

* **Malware scanning**: optional ClamAV command executes against every new upload, with a background warm-up to avoid cold-start latency. Failures block the upload and emit audit logs.
* **Content safety**: Azure Content Safety Prompt Shields (helpers/content\_safety.py) evaluates the prompt + document text via managed identity. Unsafe verdicts halt ingestion with user-facing guidance.
* **Prompt sanitization**: uploaded text is normalized, directive phrases neutralized, template markers escaped, and high-risk jailbreak cues blocked; findings are logged for later review.
* **PII detection & remediation**: Azure AI Language scans sanitized text in configurable chunks, deduplicates entities, and surfaces a remediation panel. Each finding defaults to an anonymized category/subcategory suggestion, while placeholders retain the detected term so reviewers can approve false positives (stored in a session-scoped allowlist) or supply custom redactions. Proceeding requires the scan to clear with either anonymized replacements or explicit approvals.
* **State management**: approvals persist for the current upload session and are reapplied on subsequent scans to prevent duplicate findings while still re-validating every new document.

## 3. Data Privacy and Governance

### 3.1 Data Ingestion and Minimization

* **Accepted inputs**: DOCX, PDF, JSON, or plaintext solution descriptions supplied by the user via upload. No other data sources are ingested.
* **Minimization strategy**: prompts avoid reinjecting generated content except for the *Intended Uses* list, and system instructions explicitly forbid divulging hidden rules or modifying guardrails.
* **User transparency**: the UI repeatedly states that outputs are AI-generated drafts requiring human validation before submission.
* **PII hygiene**: sanitization occurs before storage—once the remediation panel clears, only redacted text (with anonymized replacements) is persisted as the stored solution description for downstream analysis/generation.

### 3.2 Storage, Retention, and Deletion

* **Ephemeral files**: generated DOCX files are stored under rai-assessment-output/ with a unique identifier. After the user initiates a download, the app removes the file from disk. Operators should periodically purge the folder to remove orphaned outputs (automation backlog item).
* **Caching**: optional local pickle cache (./cache/completions\_cache.pkl) stores LLM responses keyed by prompt hash to save costs. Users can disable caching per run; administrators can clear the cache from the UI. No automatic TTL is currently enforced (privacy risk noted below).
* **Logging data**: blob logs capture user identity, action type, filenames, and timestamps only—generated content is not persisted in logs.
* **Stored uploads**: HTMX session state preserves the sanitized document text and approved false positives only for the active session; clearing an upload or completing generation wipes the pending PII queue.

### 3.3 Data Residency and Transfers

* All Azure resources (Key Vault, Blob Storage, Azure OpenAI) are provisioned in the target geography (e.g., Sweden Central). Network security perimeter configuration can restrict public access, ensuring data stays within Microsoft-controlled regions.

### 3.4 Privacy Risks and Mitigations

* **Residual risk**: cache persistence on the container file system may hold fragments of sensitive solution content. Mitigation options include migrating to Azure Storage with encryption, adding automated expiry, or disabling caching in production deployments.
* **Log sensitivity**: action logs contain the uploaded filename (potentially sensitive). Guidance: enforce naming policies and review log retention periods.
* **Document lifecycle**: ensure deployment scripts include scheduled cleanup of rai-assessment-output/ and /logs directories.
* **PII approvals**: session-level allowlists reduce noise but could mask future detections if the same term appears in newly uploaded files; the scan reruns for each document to mitigate, but operators should monitor for overuse of approvals.

## 4. Responsible AI Risk Mitigation

### 4.1 Intended Use, Users, and Guardrails

* Access restricted to trained RAI champions; organization-level process requires manual review prior to submission.
* The UI displays warnings before and after generation, reinforcing human oversight and accountability.
* The pre-flight “solution description analysis” highlights missing information, biases, or gaps so users can improve inputs before generation.

### 4.2 Model Selection and Configuration

* Uses the official AzureOpenAI client with deployment-scoped endpoints to guarantee responses flow through compliant infrastructure.
* Admin-only model selector presents pricing per 1K tokens, enabling cost-aware choices. Reasoning effort and verbosity are explicit controls for higher-cost models (gpt-5, o\*-series).
* Adaptive parameter builder strips unsupported parameters (temperature, top\_p, penalties) for reasoning models, reducing risk of invalid calls and inadvertent randomness.
* Fallback logic degrades gracefully (Responses API → Chat Completions) so drafts are produced even under partial service issues.

### 4.3 Human Oversight and Accountability

* Generated documents intentionally leave certain fields (e.g., sign-off sections) blank to require manual completion.
* Logs (blob + app) record who initiated generation and downloads, facilitating later audits of accountability.
* RAI champions review and edit outputs before official submission, complying with organizational review processes.

### 4.4 Transparency and Traceability

* Reasoning summaries (when available) can be displayed for administrators to review the model’s rationale.
* Cost breakdown and progress steps are surfaced in real time, supporting traceability of AI assistance.
* Outputs are clearly labelled “AI-generated draft” in the UI and within the downloaded documents.
* HTMX remediation logs include summarized threat findings (malware, prompt sanitizer, PII) so auditors can reconstruct which safeguards executed for a given upload.

### 4.5 Safety, Fairness, and Harm Mitigation

* Twelve-step template covers intended uses, stakeholders, harms assessment, and disclosure requirements, aligning with RAI template guidance and prompting explicit reflection on equity and safety.
* Prompts instruct the model not to disclose system instructions or deviate from RAI assessment scope, mitigating prompt injection.
* Azure OpenAI built-in content filtering and abuse monitoring apply to all requests.
* Cost guardrails and reasoning effort controls discourage unnecessary high-latency, high-risk model configurations.
* Pre-ingestion threat pipeline (malware, prompt shields, sanitization, PII detection) reduces the risk that hostile content reaches the LLM or that sensitive terms leak into draft outputs.

### 4.6 Responsible AI Backlog

* Add automated validation (schema checks, unit tests) for JSON outputs to reduce risk of silent errors.
* Introduce fairness and safety evaluation harnesses to spot systematic failure modes.
* Implement reasoning token visibility in the UI for every step and capture structured telemetry for RA review.

## 5. Alignment to Microsoft Responsible AI Standard (General Requirements)

| Requirement Theme | Current Controls | Gaps / Planned Actions |
| --- | --- | --- |
| Impact Assessment & Purpose | Multi-step pipeline mirrors RAI template; solution description analysis surfaces missing context; human review mandated before submission. | Formal sign-off workflow (Draft → Review → Approved) pending implementation. |
| Governance & Accountability | Entra ID auth, allow-list, admin gating, blob audit logs; reasoning summaries for admins. | Expand RBAC beyond allow-list, integrate with centralized compliance tracking. |
| Data & Privacy | Minimal data retention, cache toggle, Key Vault secrets, optional private endpoint enforcement. | Automated cache expiry, formal data retention schedule, encryption-at-rest roadmap for caches/logs. |
| Security | Managed Identity for secrets, NSP-ready deployment, rotating logs, download-once deletion. | Add rate limiting, vulnerability scanning, incident response playbook. |
| Transparency | UI warnings, downloadable logs, cost reporting, optional reasoning summary. | Provide user-facing changelog of model/config updates and link to evaluation artifacts. |
| User Experience & Accessibility | Streamlit UI with progress indicators and manual editing requirements. | Broader accessibility review (keyboard navigation, screen reader support) still pending. |
| Continuous Improvement | Memory bank tracks roadmap; backlog includes schema validation, structured telemetry, Redis cache, retry policies. | Establish regular evaluation cadence and integrate findings into change management. |

## 6. Operational Procedures

* **Change management**: Feature work tracked in memory-bank documents (activeContext.md, changelog.md). Deployment scripts (azure-container-apps/) document required configuration and service dependencies.
* **Testing**: Manual regression focused on prompt pipeline; automated testing coverage is limited (action item to expand unit and integration tests, especially for document mutations and JSON validation).
* **Incident response**: Logs (local + blob) support triage; administrators can adjust log verbosity at runtime to capture additional detail. Formal incident response drills yet to be documented.
* **Performance expectations**: Draft generation typically completes within 10–15 minutes, depending on model selection and template size. Cost outputs (from helpers/completion\_pricing.py) help teams plan budget usage.