Template Driven Analyses

Release Scope — Requirements QA Application

Gerrard Consulting Testing

01 May 2026

Contents

Executive summary

Release name: Template Driven Analyses

Application: Requirements QA Application (`* REQQA`)

Scope status: `accepted`

Release contents: 4 functional requirement(s) across 11 user story/stories, plus 40 applicable non-functional requirement(s) at application scope.

Scope created: 26 April 2026

Mission and system context

REQQA — Mission Statement

The REQQA.ai Mission Statement

REQQA operates at the intersection of requirements engineering and AI-augmented quality — applying the DeFOSPAM analytical framework to systematically identify defects in requirements and user stories before they propagate into design, code and test.

The application serves organisations developing software where the quality of input specifications directly determines the quality of output — particularly relevant as AI-assisted and agentic development approaches make specification quality the critical bottleneck.

REQQA's value proposition is improving specification quality, not accelerating careless specification.

Domain

Requirements management, specification quality assurance and AI-assisted requirements analysis for software development teams. REQQA operates at the intersection of requirements engineering and AI-augmented quality — applying the DeFOSPAM analytical framework to systematically identify defects in requirements and user stories before they propagate into design, code and test.

The application serves organisations developing software where the quality of input specifications directly determines the quality of output — particularly relevant as AI-assisted and agentic development approaches make specification quality the critical bottleneck.

The Problem REQQA.ai Is Built to Solve

Every software project begins with a description of what needs to be built. For most teams, those descriptions — requirements documents, user stories, acceptance criteria — are written quickly, reviewed informally if at all, and handed to developers with an optimistic "that should be clear enough." It usually isn't.

The cost of vague, ambiguous or incomplete specifications is enormous and well-documented: - Requirements defects are 10–100x cheaper to fix at specification time than after development begins - A single ambiguous requirement can generate weeks of rework, misaligned builds, and failed acceptance testing - The people who know what the system should do (business analysts, product owners) are often not the same people who discover what's missing (developers, testers) — and they discover it at the worst possible time

What REQQA.ai Does About It REQQA.ai provides a structured specification environment with two complementary capabilities:

1. **AI-assisted analysis** — applying the DeFOSPAM methodology to systematically examine requirements and user stories for defects, gaps, ambiguities and inconsistencies *before* they reach development
2. **AI-assisted generation** — producing well-structured first drafts of requirements from a validated application mission statement, giving teams a solid starting point rather than a blank page

What Does DEFOSPAM Stand For? DEFOSPAM is an acronym representing seven analysis steps:

- **D** - Definitions
- **F** - Features
- **O** - Outcomes
- **S** - Scenarios
- **P** - Prediction
- **A** - Ambiguity
- **M** - Missing

Each step examines the artifact (story or requirement) from a different analytical perspective.

The result is a continuous quality loop: write -> analyse -> improve -> re-analyse -> approve -> develop.

Reqqa will manage Requirements for itself

Which came first? The Reqqa application or the Reqqa requirements?

As soon as the Reqqa application is viable, the requirements for Reqqa development will be managed in Reqqa itself.

Target Users

User	Role	What REQQA Gives Them
Requirements Authors	Business analysts, product owners, technical leads	A structured environment for writing, versioning and improving requirements — with AI feedback on completeness and quality
QA Leads & Test Architects	Practitioners responsible for testability	DeFOSPAM analysis to surface ambiguities, missing scenarios and undefined terms before development begins
Team Leads & Managers	Delivery managers, scrum masters	Visibility into specification readiness, issue tracking across the portfolio, and measurable quality improvement over time
Evaluators & Facilitators	People exploring REQQA or running training sessions	A demo catalogue of realistic pre-built applications to demonstrate the full analysis workflow without manual authoring

Core Functional Areas

REQQA must provide a complete requirements management environment where users can create, edit, version and organise requirements with full audit trail. Requirements are grouped by application within a multi-tenant organisation structure, with role-based access control.

- Reqqa will manage application definitions, mission statements, requirements and associates Gherkin stories
- Reqqa will analys mission statements, requirements and Gherkin stories using a range of analysis steps, including the DeFOSPAM steps for stories, Consistency checks for Mission-Requirements, Requirement-Stories and Story-Story Consistency
- Reqqa will provide an issue-management feature to allow users to manage the backload of issues raised by the analyses or by the user community.
- Reqqa will identify all terminology requiring definition, will propose definitions and manage a glossar and index mission statements, requirements and stories to allow index searching.
- Reqqa will generate requirements based on a mission statement (for applicaitons having no requirements)
- Reqqa will maintain a catalogue of application examples that can be used to generate new applications for users to experiment with, and use for training and demonstrations.
- Reqqa will generate stories for requirements having no stories
- Reqqa will differentiate story KEY Scenarios from TEST scenarios
- Reqqa will generate a specification file (in various formats) to feed to AI Tools to auto-generate functionality
- Reqqa will maintain an update/change history for requirements ans stories
- Reqqa will generate Application Specification documentation in a range of formats for Agile and Waterfall projects

Key Constraints

Multi-tenancy — all data is strictly isolated by organisation. Users, applications, requirements and analysis results belong to exactly one organisation and must never leak across boundaries.

AI service dependency — analysis and generation features depend on external AI APIs (OpenAI, Anthropic) configured per organisation with encrypted API keys. The system must handle API failures, rate limits and timeouts gracefully without data loss.

Existing technology stack — the application is built on py4web (Python) with PyDAL ORM, MySQL database, Bulma CSS frontend and Redis Queue for background jobs. New features must work within this stack.

Specification quality over speed — REQQA's value proposition is improving specification quality, not accelerating careless specification. Features that encourage users to skip quality steps (e.g. generating requirements without validating the mission statement) undermine the product's purpose.

Deployment model — for the MVP, REQQA is deployed as a cloud-hosted, multi-tenant SaaS application accessed via standard web browser. On-premise deployment is not a requirement at this stage. A single-server deployment is acceptable at MVP scale, provided it supports the expected user load of small-to-medium software teams (up to approximately 25 concurrent users).

Data security — user authentication is session-based with HTTPS enforced throughout. AI provider API keys are encrypted at rest. Beyond these baseline controls, the MVP does not require SSO, MFA, or granular role-based access control — all authenticated users within an organisation have full access to that organisation's data.

Performance expectations — interactive page responses must complete within 3 seconds under normal conditions. AI-dependent operations (analysis, generation) are processed as background jobs and are expected to complete within defined windows: mission and story analysis within 30–120 seconds, requirements generation within 60–180 seconds. All long-running operations must provide real-time progress feedback so users are not left waiting without indication of status.

Integration — at MVP, the only external integrations are the AI provider APIs (OpenAI and Anthropic), configured per organisation. No integrations with project management tools (Jira, Azure DevOps, GitHub) or communication platforms are required at this stage. Requirements and analysis results must be exportable in structured formats (JSON, Markdown) to support manual downstream use. The application will provide import and export features to exchange data with other requirements, task management or requirements management applications on an as-needs basis.

Success Criteria

The system is working correctly when a user can create an application, write a mission statement following the guided template, have that mission analysed for completeness, generate a set of requirements from the validated mission, run DeFOSPAM analysis on the generated requirements, and iteratively improve both requirements and stories until they reach a quality level suitable for development handoff.

The specification-to-quality loop — write, analyse, improve, re-analyse — must feel natural, fast and productive. Analysis results must be actionable (not just "this is vague" but "here is what's missing and how to fix it"). Generated content must be credible enough to serve as a genuine starting point, not a toy that needs to be thrown away and rewritten.

The system must support the full lifecycle from first draft through to validated, export-ready specification — positioning REQQA as the trusted specification layer for both human development teams and future AI-assisted development pipelines.

Functional requirements

The functional requirements delivered by this release. Each is realised by one or more user stories (see the User stories section below for Gherkin scenarios).

FR-18.3.1 — Template Recommended Analyses Metadata

Format: 29148-full **Recommended Analyses:** R-D, R-F, R-C

Purpose Each requirement template carries a structured `recommended_analyses` field listing the analyses that apply by default to requirements created from that template. This is the source of truth for "which analyses fit this shape of requirement" and is the data from which per-requirement RAs are seeded (see FR-18.3.2).

Shall Statement The system shall provide a structured `recommended_analyses` field on each requirement template record, holding a comma-separated list of analyser step-codes consistent with the existing `analyses.stepcode` format, editable by a template administrator and rendered visibly on the template edit view.

Business Rules

- **BR-18.3.1.R1** Field is validated against the current set of registered analyser step-codes; unknown codes rejected at save.
- **BR-18.3.1.R2** Empty is valid; it means "no default recommendation" — requirements created from such a template start with empty RAs.
- **BR-18.3.1.R3** Changes to a template's RAs do **not** automatically propagate to existing requirements using the template. Existing requirements own their RAs once seeded — see FR-18.3.2 for the inheritance and override rules. A bulk-propagation admin action is reserved for a later release.

Acceptance Criteria

- Column `req_templates.recommended_analyses VARCHAR(255) NULL` exists.
- Template edit UI allows administrators to manage the RAs selection (checkbox or tag control) alongside other template fields.
- Template view renders the RAs visibly under a labelled section.
- Validation rejects unknown step-codes with a specific, human-readable error.

Data Requirements

- New column `req_templates.recommended_analyses VARCHAR(255) NULL`.
- Format: comma-separated step-codes, matching `analyses.stepcode` convention (for example `'R-D,R-F,R-C'`).

Dependencies

- Existing analyser step-code registry.
- FR-18.3.2 (the consumer — seeds per-requirement RAs from the template).
- FR-18.3.5 (the initial-population migration that populates RAs on existing templates for the REQQA app).

Verification

- Inspection of schema change per the `/pydal-migrate` procedure.
- Demonstration: template edit shows and saves RAs; invalid codes rejected.

Notes **Wave 1 (foundation).** Delivered together with FR-18.3.2 so the two- layer RA model is usable in a single release.

FR-18.3.2 — Requirement Recommended Analyses

Format: 29148-full **Recommended Analyses:** R-D, R-F, R-C

Purpose Each requirement carries its own `recommended_analyses` (RAs) field which is seeded from its template's RAs at creation or at template-apply, and can be overridden per requirement thereafter. Inheriting from the template (FR-18.3.1) avoids re-keying the same list for every requirement while preserving author control on a case-by-case basis.

Shall Statement The system shall provide a structured `recommended_analyses` field on each requirement record, seeded from the requirement's template (FR-18.3.1) on creation or on template application, editable by a requirement author, and rendered visibly on the requirement view.

Inheritance and Override Rules

- **BR-18.3.2.R1** On **create with a template** -> requirement RAs are copied from the template's RAs at save time. The author may then edit.
- **BR-18.3.2.R2** On **change of template** for an existing requirement -> the system prompts the author with three choices: **Replace** (overwrite with the new template's RAs), **Keep** (retain current RAs), **Merge** (union of current and new-template RAs). **Default action: Keep** — changing the template must never silently destroy author overrides.
- **BR-18.3.2.R3** On **template edit** (a template's own RAs change) -> no automatic propagation to existing requirements using that template. Each existing requirement owns its RAs. A bulk-propagation admin action is reserved for a later release.
- **BR-18.3.2.R4** On **removal of a template** from a requirement -> requirement RAs are preserved as-is. The requirement continues to carry its existing RAs.
- **BR-18.3.2.R5** On **no template + new requirement** -> RAs is empty. The kickoff UI (FR-18.3.3) falls back to showing all analyses unselected.

Business Rules

- **BR-18.3.2.R6** Field applies to requirements only — not stories, personas, or definitions.
- **BR-18.3.2.R7** Values are validated against the registered analyser step-codes; unknown codes rejected with a specific message.
- **BR-18.3.2.R8** Empty is a valid value (see BR-18.3.2.R5).
- **BR-18.3.2.R9** Retro-placeholder requirements (`origin='retro-placeholder'`) may leave RAs empty and are typically excluded from analysis runs by default.

Acceptance Criteria

- Column `requirements.recommended_analyses VARCHAR(255) NULL` exists.
- Same column exists on `requirements_history`.
- On create with a template -> requirement RAs equals the template's RAs (verified with a test fixture).
- On template change with existing non-empty RAs -> user is prompted; the default choice is Keep.
- Requirement edit UI shows and persists the selection.
- Requirement view renders the RAs under a visible **Recommended Analyses** heading.
- Validation rejects unknown step-codes.

Data Requirements

- New column `requirements.recommended_analyses VARCHAR(255) NULL`.
- Same column on `requirements_history`.

- Same format as `analyses.stepcode` (comma-separated step-codes).

Dependencies

- FR-18.3.1 (template RAs — the inheritance source).
- FR-18.3.3 (analysis kickoff UI — the primary consumer).
- Existing analyser step-code registry.
- Existing template-application workflow in the requirement edit UI.

Verification

- Inspection of schema change.
- Demonstration: create with template -> RAs seeded; change template -> prompt shown and Keep is default; author override persists.
- Test: an analysis run launched from a requirement with non-empty RAs pre-selects exactly those analyses in the kickoff UI.

Notes **Wave 1 (foundation).** Delivered with FR-18.3.1 and FR-18.3.5 as a single release so the RA model is functional in the REQQA app from day one.

FR-18.3.3 — Analysis Kickoff Pre-Selection

Format: shall-light **Recommended Analyses:** R-F, R-C

Background When an analyst kicks off an analysis run from a requirement, the analysis selector checkbox list is pre-ticked with the requirement's `recommended_analyses` metadata. The analyst can add or remove selections before running.

Shall Statement The system shall pre-select analyses on the analysis-kickoff UI using the requirement's `recommended_analyses` metadata (FR-18.3.2), allowing the operator to modify the selection before the run begins.

Acceptance Criteria

- Checkbox list is pre-populated from the requirement's `recommended_analyses`.
- Operator can tick or untick freely.
- Final selection — not the metadata — is what gets executed and logged on the `analyses` row.
- If `recommended_analyses` is empty, no analyses are pre-ticked.

Dependencies FR-18.3.2 (the data source — requirement RAs).

Notes Small UI change; consider folding into FR-18.3.2's delivery if it is trivially wired together.

FR-18.3.5 — Initial Gold-Template RA Population

Format: shall-light **Recommended Analyses:** *n/a — one-off migration requirement*

Background FR-18.3.1 and FR-18.3.2 introduce `recommended_analyses` on templates and requirements but deliver the fields empty. The "Gold" templates — the canonical master from which every org's local `req_templates` rows are derived — live in the repository at `static/templates/reqs_templates/` as twenty-seven `.txt` files (functional, NFR, TR, and ISO 29148 classes). Each file carries YAML-style front-matter with `name:` and `tags:` keys.

This requirement is the one-off CLI migration that brings RAs into existence across the live system in two coordinated phases:

1. **Edit the Gold `.txt` files** — add a `recommended_analyses:` key to each file's front-matter, alongside `name:` and `tags:`. Comma- separated step-codes from the analyser registry (`modules/statusutils.py STEP_CODES`).
2. **Phase `templates`** — for every row in `req_templates` whose `name` matches a gold file's `name` key, overwrite `recommended_analyses` with the gold value. Per-template commit.
3. **Phase `requirements`** — for every requirement with `template_id` set and `recommended_analyses` empty, copy the parent template's `recommended_analyses` into the requirement's `recommended_analyses`. Per-record commit. Skips listed below.

Shall Statement The system shall be initialised, on delivery of FR-18.3.1 and FR-18.3.2, by running a CLI migration script that (a) populates `req_templates.recommended_analyses` from the gold `.txt` files for every org, and (b) seeds `requirements.recommended_analyses` from the populated template rows where the requirement's RAs is empty.

Migration Mechanism (locked 2026-04-27)

- **Invocation.** Standalone Python script at `documentation/migrations/0.X/populate_template_ras.py`. Argument `--phase` accepts `templates` | `requirements` | `all` (default `all`).
- **Template phase — overwrite always.** Templates are matched by `name`. Existing `recommended_analyses` on a template row is overwritten with the gold value unconditionally.
- **Requirement phase — seed only if empty.** Requirements with an existing non-empty `recommended_analyses` are skipped (status `already-populated`).
- **Atomicity — per-record commit.** No global transaction. A failure on one row does not roll back successful prior rows. The next unprocessed row is attempted.
- **Recovery — re-run, no rollback.** The script is idempotent: rerun reaches a consistent state regardless of any partial prior state. No pre-migration backup table or in-script rollback logic is required.
- **Validation timing — per row.** No pre-check across the whole corpus; failures surface as the script encounters them and are recorded in the report.
- **Step-code registry validation deferred.** Gold files are trusted to contain only step-codes registered in `STEP_CODES`. Future hardening to validate against the registry on every run is captured as a backlog item, not v1 scope.

Skip Categorisation (fixed precedence, requirement phase) When a requirement is eligible for multiple skip reasons, the first match in this order wins (exactly one reason recorded per row):

1. `retro-placeholder` — `requirements.origin = 'retro-placeholder'`
2. `no-template` — `requirements.template_id IS NULL`
3. `already-populated` — `requirements.recommended_analyses` is non-empty
4. `template-not-yet-populated` — the requirement's template has empty `recommended_analyses` (run `--phase=templates` first)

Migration Report A Markdown report is written at `documentation/migrations/0.X/populate_template_ras_report_<` on every run. The same content is printed to stdout (Claude Code console). The report contains:

- A header summary with counts per `status` and per phase
- One row per processed template and per processed requirement
- Each row's `status` is exactly one of `applied`, `skipped`, `failed`, `unchanged`, with a `reason` for non-`applied` rows

Scope

- **In scope:** All orgs and all apps in the current system. The Gold files are edited in place; the cascade and seeding apply system-wide. Outside REQQA's own org, all data is test fixtures owned by the project team — awareness of the change is recorded and considered acceptable.

- **Excluded — requirements without a template.** They cannot be seeded (no source). Captured as `no-template` skip in the report.
- **Excluded — retro-placeholder requirements** (`origin = 'retro-placeholder'`). Captured as `retro-placeholder` skip.
- **Excluded — registry validation in v1.** Future hardening item.
- **Out of scope for v1:** automatic re-seeding on template edit after initial population (deferred to a future bulk-propagation admin action).

Acceptance Criteria

- Every `.txt` file in `static/templates/reqs_templates/` carries a reviewed `recommended_analyses:` key in front-matter, or an explicit empty value with a recorded "no RAs apply" decision.
- Every row in `req_templates` (across all orgs) has its `recommended_analyses` set to match the corresponding Gold file's value after `--phase=templates` runs.
- Every requirement (across all apps) whose template carries non-empty RAs and whose own RAs is empty has its RAs seeded after `--phase=requirements` runs, except rows where `origin = 'retro-placeholder'`.
- Migration script lives at `documentation/migrations/0.X/`.
- Markdown report is produced on every run, mirrored to stdout.
- Re-running the script produces a consistent end state regardless of prior partial state.

Dependencies

- FR-18.3.1 (template RAs column must exist first).
- FR-18.3.2 (requirement RAs column and inheritance rules must exist first).

Notes **Wave 1 (foundation)** — delivered together with FR-18.3.1 and FR-18.3.2. Untemplated requirements are an existing data-hygiene issue tracked separately for later cleanup.

Applicable non-functional requirements

The non-functional requirements applicable to the application containing this release. These constraints — performance, security, reliability, safety, and other quality attributes — apply to every release and must be satisfied by the capability delivered under this scope.

FR-00 — Functional Requirements

Priority: **High**

FR-02 — Requirements Authoring and Organization

Priority: **High**

The system shall enable users to create, edit, version, and organize requirements within applications, with each requirement supporting rich text content, priority assignment, and status tracking. Requirements must be grouped logically within their parent application and support hierarchical organization where appropriate. The system shall maintain a complete audit trail showing who created or modified each requirement, when changes occurred, and what content changed. Users must be able to link requirements to related requirements, mark dependencies, and associate requirements with one or more user stories. The system shall prevent deletion of requirements that have associated stories or open issues, requiring users to first resolve dependencies.

FR-03 — User Story Management with Gherkin Support

Priority: **High**

The system shall provide comprehensive user story management where each story is written in Gherkin format with Given-When-Then structure. Users must be able to create stories manually or generate them from requirements using AI assistance. The system shall differentiate between KEY scenarios that define core functionality and TEST scenarios that cover edge cases, variations, and error conditions. Each story must be linked to at least one parent requirement and support multiple scenarios within a single story. The system shall validate Gherkin syntax and highlight structural issues such as missing Given clauses or malformed Then statements. Story versioning must track all changes with full audit history.

FR-04 — DeFOSPAM Analysis for Stories and Requirements

Priority: **High**

The system shall execute DeFOSPAM analysis on user stories and requirements, applying all seven analytical steps: Definitions, Features, Outcomes, Scenarios, Prediction, Ambiguity, and Missing. Each analysis step must examine the artifact from its specific perspective and generate actionable findings with severity ratings and suggested remediation. Analysis must be performed as a background job with real-time progress updates visible to the user. Results must be persisted and associated with the specific version of the artifact analyzed, allowing users to compare analysis results across versions. The system shall support re-analysis after modifications, automatically detecting when an artifact has changed since its last analysis and prompting users to re-run quality checks.

FR-05 — Consistency Analysis Across Artifacts

Priority: **High**

The system shall perform consistency analysis across related artifacts, checking mission-to-requirements alignment, requirements-to-stories traceability, and story-to-story coherence. Mission-to-requirements consistency analysis must verify that generated or authored requirements adequately address all aspects of the mission statement, identifying gaps where mission objectives lack corresponding requirements. Requirements-to-stories analysis must ensure that each requirement has sufficient story coverage and that stories do not introduce functionality beyond their parent requirements' scope. Story-to-story analysis must detect contradictions,

overlapping scenarios, and inconsistent terminology across stories within the same application. All consistency findings must be surfaced as issues in the issue management system.

FR-06 — AI-Assisted Requirements Generation

Priority: **High**

The system shall generate a complete first-draft set of requirements from a validated mission statement using configured AI providers. Generation must only be permitted after the mission statement has been analyzed and all critical issues resolved, enforcing the quality-first workflow. The generation process must run as a background job with progress tracking, typically completing within 60-180 seconds. Generated requirements must be structured with clear titles, detailed content prose, assigned types (functional, non-functional, technical), and suggested priorities. Users must be able to review generated requirements in a staging area before accepting them into the application, with options to edit, reject, or regenerate individual requirements. The system shall track which requirements were AI-generated versus manually authored.

FR-07 — Issue Management and Resolution Workflow

Priority: **Medium**

The system shall provide comprehensive issue management for tracking defects, ambiguities, and improvement suggestions identified through analysis or raised manually by users. Each issue must be linked to the specific artifact (mission statement, requirement, or story) and artifact version where it was detected. Issues must support severity classification, status tracking (open, in progress, resolved, closed), assignment to users, and resolution notes. The system shall automatically create issues from analysis findings but allow users to suppress false positives or mark findings as accepted risks. Users must be able to filter and sort issues by application, artifact type, severity, status, and age. The system shall prevent artifacts with open critical issues from being marked as approved for development.

FR-08 — Terminology Management and Glossary

Priority: **Medium**

The system shall automatically identify domain terminology requiring definition across all artifacts within an application, using AI analysis to detect technical terms, acronyms, and domain-specific concepts. For each identified term, the system shall propose definitions based on context and usage patterns, which users can accept, modify, or replace. The glossary must be maintained at the application level, ensuring consistent terminology across all requirements and stories. Terms in artifact content must be automatically linked to their glossary definitions, and the system shall detect when the same concept is referred to using different terms, suggesting standardization. The glossary must support full-text search and provide an index showing where each term is used across the application's artifacts.

FR-09 — Application Catalogue and Template Management

Priority: **Medium**

The system shall maintain a catalogue of pre-built example applications that users can browse, preview, and instantiate into their organization's workspace for training, demonstration, or experimentation purposes. Each catalogue entry must include a complete application with mission statement, requirements, stories, and sample analysis results. Users must be able to create new applications from catalogue templates, which copies all artifacts while maintaining clear provenance showing the template source. The system shall support both public catalogue entries available to all organizations and organization-specific private templates. Catalogue applications must be maintained separately from user workspaces and must not count against organization storage or usage limits.

FR-10 — Specification Export and Documentation Generation

Priority: **Medium**

The system shall generate comprehensive specification documentation in multiple formats suitable for both Agile and Waterfall development methodologies. Export formats must include JSON for programmatic consumption, Markdown for documentation systems, and formatted PDF for formal review processes. Generated specifications must include the mission statement, all requirements organized by type and priority, associated user stories with full Gherkin scenarios, glossary terms, and a traceability matrix linking requirements to stories. The system shall support filtered exports where users can select specific requirements or stories to include, and must embed metadata such as version numbers, export date, and approval status. Exports must be suitable for feeding to AI development tools as specification input.

FR-11 — Multi-Tenant Organization Management

Priority: **High**

The system shall enforce strict multi-tenant data isolation where all applications, requirements, stories, issues, and analysis results belong to exactly one organization and are never visible across organizational boundaries. Each organization must have its own encrypted AI provider API keys for OpenAI and Anthropic services, configured by organization administrators. The system shall support user invitation and management within organizations, with session-based authentication and HTTPS enforcement for all connections. Organization administrators must be able to view usage metrics including API consumption, storage utilization, and active user counts. The system shall prevent any database queries or API responses from leaking data across organizational boundaries, enforcing isolation at the data access layer.

FR-12 — Background Job Processing and Progress Tracking

Priority: **High**

The system shall process all AI-dependent operations (mission analysis, requirements generation, story analysis, consistency checks) as background jobs using Redis Queue, ensuring that long-running operations do not block the user interface. Each background job must provide real-time progress updates showing current step, estimated completion time, and any interim results. Users must be able to navigate away from the page that initiated a job and return later to view results, with the system preserving job status and providing notifications when jobs complete. The system shall handle AI API failures gracefully by retrying transient errors, logging permanent failures, and allowing users to manually retry failed jobs without losing context or previous work.

FR-17 — Help System (GUIDE)

Format: 29148-full **Recommended Analyses:** R-D (Defect), R-F (Format/structure), R-C (Consistency)

Purpose Define an end-to-end, in-app Help System for REQQA — codename **GUIDE** — whose content, indexing, and lifecycle are maintained primarily by Claude Code development skills on phase entry and phase exit, supplemented by human authoring where appropriate.

Scope GUIDE covers **user-facing help** only. Operator and administrator documentation (deployment, backup, service management) is out of scope and is handled separately via markdown documents in the repository.

GUIDE is **global by default** (one help corpus shared across all tenant organisations) with a cheap architectural provision for **tenant-specific overrides** in future releases — delivered by a nullable `orgid` on help-page records (NULL = global default, non-NULL = org override).

Actors / Stakeholders

- **End users** of REQQA (analysts, builders, product owners) — primary consumers.
- **Claude Code development skills** — primary authors and maintainers; they read existing help on phase entry and write/update help on phase exit.
- **Paul (product owner)** — reviews candidate FAQ entries, approves retro- documentation, owns editorial standards.
- **Prospect visitors** at `reqqa.ai` — consume a *public* subset of the help corpus (feature overview, release notes flagged public), not the full manual.

Definitions

- **Help page** — a piece of help content keyed to a template (user-visible page) or a template-plus-panel identifier (embedded sub-view).
- **How-to guide** — task-oriented article that spans multiple pages.
- **Help glossary** — scoped view over the existing `definitions` / `termindex` tables; reuses the project glossary engine.
- **GUIDE skills** — two Claude Code skills (phase-entry and phase-exit) that read, gap-detect, write, and re-index help content around a development phase.

Principles / Help-Wide Rules

- **HR-17.P1 Template-keyed mapping.** Help entries are keyed to user-visible templates (and optionally to a panel identifier within a template), not to py4web routes or controller actions. Rationale: users navigate pages, not routes.
- **HR-17.P2 Inclusion criterion.** Every template that renders a user-reachable *primary or sub-view* has a GUIDE entry. Partial/fragment templates rendered inside others and pure authentication-flow templates (login, logout) are excluded.
- **HR-17.P3 Authorship model.** Primary authors are Claude Code development skills executing the GUIDE phase-entry and phase-exit protocols (FR-17.16). Human override is always available via the standard edit UI.
- **HR-17.P4 Content storage.** DB-backed (new `help_pages` table) with a markdown import/export round-trip for bulk authoring. Glossary reuses the existing `definitions` / `termindex` tables by adding a scope marker.
- **HR-17.P5 Visibility.** Default visibility is **authenticated users only**. A per-entry visibility flag (`public` | `authenticated`) enables selected content (e.g. release notes, feature overview) to surface on the guest homepage and at `reqqa.ai`.
- **HR-17.P6 Role-awareness.** Help entries carry role visibility so that admin-only topics are hidden from ordinary users.
- **HR-17.P7 Staleness model.** Each help page carries a `last_touched_at` timestamp and a `last_touched_by_phase` marker. A page is flagged **stale** if the template it documents has changed since `last_touched_at`, or if the staleness window (default 180 days; subject to tuning) has elapsed without review.
- **HR-17.P8 No user-data in help content.** Help pages are generic product documentation. They must not reference tenant data, user names, or specific customer scenarios.

System Interfaces

Interface	Type	Direction	Notes
layout.html help button	UI	Read	Fetches help for current template key
Help pages UI	UI	Read/Write	Viewer + admin authoring surface
GUIDE phase-entry skill	Process	Read	Inspects help gaps before a phase
GUIDE phase-exit skill	Process	Write	Updates pages and re-indexes glossary
Guest homepage	UI	Read	Renders `visibility=public` entries
Existing glossary (`definitions`, `termindex`)	Internal	Read/Write	Shared engine, new `scope` column
Analyses engine	Internal	Read	For help-specific analysis runs

Data Requirements

- New table `help_pages`: `id, orgid NULL, template_key, panel_key NULL, title, content, visibility ENUM('public','authenticated'), role_visibility, last_touched_at, last_touched_by_phase, status ENUM('stub','draft','published','stale'), created, created_by, updated, updated_by`.
- Extend `definitions` / `termindex` with a `scope` column (`project` | `help` | `both`) — see FR-17.3.
- New table `help_feedback`: per-page thumbs and optional comment (see FR-17.11).
- Search analytics log: stored queries and result counts (see FR-17.12).

Detailed schemas live in the relevant child requirements.

Acceptance Criteria (parent-level)

- Every user-reachable primary or sub-view template in REQQA has at least a `stub` help page, rendered when the layout.html help button is clicked.
- The help corpus is searchable in under 1 second for typical queries (provisional; see FR-17.4).
- Help glossary terms share a single editing surface with project glossary terms, distinguished by `scope`.
- The GUIDE phase-exit skill successfully updates or creates help pages for every template touched in a phase, and re-indexes the glossary.
- A `status='stale'` page surfaces in an authoring worklist for review.

Quality Attributes / NFRs

- **Performance** — help button -> page render in under 500 ms (provisional).
- **Search** — unified search returns results in under 1 s for P95 under typical load (provisional).
- **Coverage** — 100 % of user-reachable primary/sub-view templates have at least a stub; measured by automated template-vs-help-page audit.
- **Multi-tenant isolation** — org overrides must never leak across organisations when tenant-override editing is later enabled.

Dependencies

- Existing `definitions` / `termindex` tables and their editor/indexer.
- Existing analyses engine (for help-specific analysis runs).
- Claude Code skills infrastructure (global skills under `~/.claude/commands/`).

Out of Scope

- Operator / administrator documentation (handled via repository markdown).
- Multi-language / i18n — English only for the foreseeable future.
- In-product tours / guided walkthroughs beyond a simple onboarding flow (FR-17.6).
- Automated translation of help content.
- Help content customised per end-user (as distinct from per-organisation).

Verification

- Inspection — template coverage audit (template list vs `help_pages`).
- Demonstration — help button on representative pages; guest homepage public entries; glossary editor showing help-scoped terms.
- Test — search latency benchmarks; phase-entry skill reading; phase-exit skill writing and re-indexing.

Open Issues

- Exact `staleness` window default — 180 days is a working proposal.
- Whether tenant-override editing ships in a subsequent wave or is left as architectural provision only.
- AI help assistant (FR-17.10) is flagged **experimental / Wave 4**; decision on whether to build it at all deferred until evidence from search analytics justifies it.

Decomposition The parent decomposes into sixteen children — numbered FR-17.1 to FR-17.16 — covering context-sensitive pages, how-tos, glossary, search, inline help, onboarding, release notes, FAQ, keyboard shortcuts, AI assistant, feedback, analytics, role visibility, screenshot freshness, PDF export, and the phase-entry/phase-exit authoring skills. Children are grouped into four delivery waves (foundation, usability, feedback loop, polish/marketing) and will be refined as delivery approaches.

FR-17.1 — Context-Sensitive Help Pages

Format: 29148-full **Recommended Analyses:** R-D, R-F, R-C

Background Every user-visible template in REQQA should have a help entry reachable from a single help button rendered by layout.html. The button "knows" the current context and resolves help by `(template_key, active_panel_key)` — where the active panel key is the currently active tab, modal, or sub-view (or `NULL` where the page has no panels).

Shall Statement The system shall render a help page matching the current template and, where applicable, the currently active panel within that template, when the user activates the help control in the page layout.

Help Button Placement Rule

- There is **one** help button per page, placed in a fixed position by `layout.html`. No per-panel buttons. Users always know where to click.
- The button resolves help by `(template_key, active_panel_key)`.
- Resolution order:
  1. Exact match on `(template_key, active_panel_key)` -> panel help renders, with a "See also: page help for X" link at the top.
  2. If no panel is active, or the panel has no dedicated help -> fall back to `(template_key, NULL)` — page help renders.
  3. If no help exists at either level -> user sees a "stub — help not yet written" message, not a 404; the template key is logged for gap- detection (FR-17.12 analytics).

Related How-Tos Rule (back-references)

- Every context page's help view renders a **Related how-tos** section populated from the back-reference index derived from FR-17.2.
- Back-references are **not manually authored** — they are derived from the explicit forward links in how-to content. The index updates when how-tos are saved, so context pages never carry stale or orphaned references.
- A page with no referencing how-tos omits the section entirely.

Acceptance Criteria

- A single help button appears on every page served by the main layout.
- Clicking resolves via the order above; every user-reachable primary or sub-view renders *something* (real content, fallback, or stub message).
- The panel-level view carries an explicit link back to page-level help.
- Context pages render a "Related how-tos" section when the back-reference index contains entries.
- Page renders in under 500 ms for typical content.

Dependencies FR-17 parent; `help_pages` table; FR-17.2 (forward links authored in how-to content); FR-17.12 (for logging gap queries).

Notes Inclusion criterion defined at parent (HR-17.P2). Panels with genuinely distinct tasks warrant their own `(template_key, panel_key)` help entry; panels whose help overlaps the page don't need one. FR-17.5 (inline help indicators) covers micro-level field/control help and is complementary. **Wave 1 (foundation).**

FR-17.10 — AI Help Assistant

Format: shall-light — EXPERIMENTAL / WAVE 4 / BUILD-ONLY-IF-JUSTIFIED **Recommended Analyses:** R-D, R-F

Background An AI assistant that answers natural-language questions scoped to the REQQA help corpus. Included for completeness of the vision but flagged as **experimental**. The FAQ-mining loop (FR-17.8) does not depend on this — it runs off search analytics and feedback. Build only if user feedback shows unified search (FR-17.4) is insufficient.

Shall Statement If built, the system shall provide an AI help assistant that answers user questions grounded in the help corpus, using the enquiring organisation's configured AI credentials (org pays for its own help queries) and logging each call to `gptlog` alongside other AI calls.

Acceptance Criteria

- Assistant responses cite the help pages they drew from.
- Out-of-corpus refusal is explicit: the assistant declines to answer questions outside the help domain.
- Token usage appears in `gptlog` and the org's usage reports.
- Question logs feed the FAQ candidate queue (FR-17.8) without auto-publishing answers.

Dependencies FR-17 parent; existing `gptlog` and per-org AI configuration.

Notes **Wave 4 / experimental.** A design decision is required on free-quota vs org-pays-always; current assumption is org-pays-always.

FR-17.11 — Help Page Feedback

Format: shall-light **Recommended Analyses:** R-F, R-C

Background Each help page carries a simple 'was this helpful?' prompt (thumbs / 3-point or 5-point rating) with an optional free-text comment. Feedback drives prioritisation of help improvements via the analytics view (FR-17.12) and FAQ candidate queue (FR-17.8).

Shall Statement The system shall capture per-help-page feedback — a numeric rating and an optional comment — and associate it with the user (authenticated) or session (guest).

Acceptance Criteria

- Feedback control appears on every help page.
- Submission is asynchronous; no page reload required.
- Ratings and comments are queryable for administrative review.
- Spam/abuse protection appropriate to the authenticated-only default.

Dependencies FR-17.1, FR-17.12.

Notes **Wave 3 (feedback loop).**

FR-17.12 — Help Analytics

Format: shall-light **Recommended Analyses:** R-F, R-D

Background Log search queries, result counts, page views, and (via FR-17.11) feedback. Provides the basis for FAQ candidate detection (FR-17.8), stale-page identification, and coverage reports.

Shall Statement The system shall log help search queries with result counts, help page views, and feedback events; and provide an administrator dashboard summarising top queries, zero-result queries, most- and least-viewed pages, and lowest-rated pages.

Acceptance Criteria

- Log entries are queryable for the preceding 90 days at minimum.
- Zero-result queries are flagged prominently for FAQ-gap detection.
- Dashboard respects tenant isolation if tenant-override content is enabled.

Dependencies FR-17.4, FR-17.11.

Notes **Wave 3 (feedback loop).**

FR-17.13 — Role-Aware Help Visibility

Format: shall-light **Recommended Analyses:** R-C

Background Admin-only topics (tenant settings, billing, AI configuration) should be hidden from ordinary users to reduce noise and avoid confusion.

Shall Statement The system shall filter help content by the viewer's role(s) so that admin-only help pages are not rendered, indexed, or searchable for users lacking the required role.

Acceptance Criteria

- Each help page declares a role-visibility set.
- Search, glossary, FAQ, and context help all respect role visibility consistently.
- Authoring UI warns if an admin-only page is referenced by a non-admin page.

Dependencies FR-17 parent (HR-17.P6).

Notes **Wave 2 (usability).**

FR-17.14 — Screenshot and Diagram Freshness

Format: shall-light **Recommended Analyses:** R-F

Background Help pages inevitably include screenshots and diagrams. These become stale faster than text. A lightweight per-image version stamp and a manual review flag help prevent misleading users.

Shall Statement The system shall track a version stamp and last-reviewed date for each embedded screenshot or diagram in a help page, and surface images whose stamp is older than the configured freshness window for review.

Acceptance Criteria

- Images carry metadata: version, last reviewed, reviewer.
- Overdue images appear in the authoring worklist.
- Bulk-replace by version is possible.

Dependencies FR-17.1, FR-17.16 (the phase-exit skill updates stamps).

Notes **Wave 3 (feedback loop).**

FR-17.15 — Print and PDF Manual Export

Format: shall-light **Recommended Analyses:** R-F

Background Some users want a single PDF user manual. Generated from the same help corpus via pandoc (the same toolchain used elsewhere in REQQA for document export).

Shall Statement The system shall export a single PDF user manual assembled from published help pages, how-to guides, and help glossary entries, ordered by a configured table of contents.

Acceptance Criteria

- Export is available to authenticated users.
- Generated PDF respects role-aware visibility (admin vs user manual as separate exports).
- Section ordering is configurable by an admin.
- `pandoc/pdflatex` toolchain reuses the pattern already in use for scope reports.

Dependencies FR-17.1, FR-17.2, FR-17.3, FR-17.13. Operational note: subprocess-fork SIGCHLD reset required (see existing REQQA pattern for pandoc).

Notes **Wave 4 (polish).**

FR-17.16 — Phase-Entry and Phase-Exit Authoring Skills

Format: 29148-full **Recommended Analyses:** R-D, R-F, R-C

Background GUIDE is maintained primarily by Claude Code skills running on phase boundaries. On phase entry, a skill reads help for the templates in scope (giving Claude context) and identifies help gaps. On phase exit, a skill updates or creates help for templates touched in the phase, updates `last_touched_at` and `last_touched_by_phase`, refreshes the glossary index, and flags stale screenshots.

Shall Statement The system shall provide two Claude Code skills — **guide-entry** and **guide-exit** — installed globally under `~/.claude/commands/`, with a well-defined contract against the `help_pages` table, the glossary tables, and the search/analytics indexes.

Acceptance Criteria

- `guide-entry`: given a phase scope (set of templates), reads matching help pages, surfaces templates with no help, and outputs a reading list + gap list for Claude's context.
- `guide-exit`: given the set of templates actually touched in the phase, updates or creates help pages, sets `last_touched_at`, `last_touched_by_phase`, triggers glossary re-indexing, and flags stale screenshots.
- Both skills leave an audit trail in a `help_activity` log.
- Both skills are idempotent and safe to re-run.
- Human review of machine-authored pages is supported via the standard help-page edit UI with a `reviewed_by_human` flag.

Dependencies FR-17.1 (help_pages), FR-17.3 (glossary), FR-17.11, FR-17.12, FR-17.14. Depends on the Claude Code skills infrastructure.

Notes Load-bearing: without this, GUIDE is not sustainable. **Wave 1 (foundation).** Skills live in `~/.claude/commands/` per the global/framework convention.

FR-17.2 — How-To and Task Guides

Format: shall-light **Recommended Analyses:** R-F, R-C

Background Some user goals span multiple pages (e.g. "run an analysis" covers the requirement list, the analysis kickoff page, and the results view). These cross-cutting guides are authored as articles separate from per-page context help. A how-to is discovered via unified search (FR-17.4), via the help index, and via **back-references rendered on the context pages it mentions** (see FR-17.1).

Shall Statement The system shall provide a library of task-oriented how-to guides, each explicitly linking to the context pages it references, with the reverse relationship (context page -> how-tos that reference it) automatically derived and rendered on the referenced pages.

Bidirectional Linkage Rule

- **Forward links** are authored explicitly in how-to content. The author writes anchor references to context pages by template key (e.g. `[reqsList](help://template/reqsList)` or an equivalent canonical form).
- **Back-references** are derived, not authored. When a how-to is saved, the system extracts its forward links and populates a `help_references` index keyed by the referenced template. Context pages (FR-17.1) render a "Related how-tos" section from this index.
- The back-reference index is the single source of truth for reverse relationships. Authors never maintain back-links by hand; stale and orphaned references cannot occur.
- Unpublished or draft how-tos do not contribute to the back-reference index seen by ordinary users.

Acceptance Criteria

- How-to guides are discoverable through unified search (FR-17.4) and via the help index.
- Context pages automatically show a "Related how-tos" section for every page they reference.
- The reverse index refreshes on save; no manual maintenance step.
- How-to entries carry the same staleness metadata and version history as context pages.
- A how-to whose forward links contain unknown template keys is flagged in the authoring UI.

Dependencies FR-17.1 (consumer of the back-reference index), FR-17.4 (unified search), new `help_references` table or equivalent index mechanism.

Notes The `help_references` table is the smallest implementation (source how-to id, target template key, extracted on save). Alternative: inline anchor tags in content parsed on read — simpler schema but higher read cost. Decision deferred to Wave 1 design. **Wave 2 (usability).**

FR-17.3 — Help Glossary

Format: 29148-full **Recommended Analyses:** R-D, R-F, R-C

Background REQQA already has a glossary engine (tables `definitions` and `termindex`) used for project-scoped requirements terminology. The help system needs a glossary for app-level terms (Release, Scope, Phase in REQQA's sense). Rather than build a parallel engine, the existing tables are extended with a `scope` column so one editor and one indexer serve both.

Shall Statement The system shall provide a help glossary, persisted in the existing `definitions` and `termindex` tables, distinguished from project glossary entries by a `scope` column with values `project`, `help`, or `both`.

Acceptance Criteria

- Existing glossary editor UI works on help-scoped entries without code duplication.
- Help glossary terms appear in the help index and are searchable via unified search (FR-17.4).
- In-page terms in help content link to their glossary entry via `termindex`.
- Adding a `scope` column does not break existing project-scoped queries.

Dependencies Existing `definitions` and `termindex` schema; FR-17.4 (unified search).

Notes Schema change requires a migration cycle (see `/pydal-migrate`). **Wave 1 (foundation).**

FR-17.4 — Unified Help Search

Format: shall-light **Recommended Analyses:** R-D, R-F

Background A single search surface spans help pages, how-to guides, FAQ entries, and help-scoped glossary terms. Results are grouped by type.

Shall Statement The system shall provide a unified search across all help content types and return results grouped by type within 1 second at P95 under typical load.

Acceptance Criteria

- Single search input in the help UI.
- Results grouped: Pages / How-tos / FAQ / Glossary.
- Zero-result queries are logged for FAQ-gap detection (see FR-17.12).
- Search respects visibility and role rules (FR-17.13).

Dependencies FR-17.1, FR-17.2, FR-17.3, FR-17.8, FR-17.12

Notes **Wave 2 (usability).**

FR-17.5 — Inline Help Indicators

Format: shall-light **Recommended Analyses:** R-F, R-C

Background Info-icons and tooltips next to form fields and complex controls give users a one-line explanation plus a deep link into the relevant help page. This is help delivered where the question arises, not on a separate surface. Inline help is **opt-in per control**, not universal — adding a marker to every button becomes noise that undermines signal.

Shall Statement The system shall render info-icons and tooltips at template-declared UI affordances, each carrying a short explanation and a deep link to the relevant context-sensitive help page, with the set of controls that warrant a marker governed by the Opt-In Rule below.

Opt-In Rule Inline help is added to a control only if at least one of the following is true:

- The label alone is insufficient to know what to enter or what will happen.
- There is a non-obvious constraint (format, length, side-effect, consequence).
- The control interacts with others in a way that is not visible on screen.
- The control is rarely used and users routinely forget what it does.

Inline help is **not** added to routine controls where it would become noise: Save / Cancel / Close buttons, form fields with self-explanatory labels, routine navigation elements, or controls whose behaviour is entirely conventional.

Consistency Rule Once the Opt-In Rule is adopted, new form-building work must follow it so the product does not drift back to ad-hoc tooltips. Departures from the rule require an explicit decision recorded with the change.

Existing Tooltip Audit (scope-of-delivery) The current REQQA template set carries ad-hoc inline hints (HTML `title` attributes, `fa-info-circle` / `fa-question` icons, `has-tooltip` classes) across approximately fifteen templates with no consistent rule. Delivery of FR-17.5 must include an audit of every such existing marker:

- Markers that meet the Opt-In Rule are migrated to the 17.5 pattern with a deep link to a real help entry keyed by `(template_key, control_id)`.
- Markers that do not meet the rule are deliberately removed as noise.
- The audit leaves the template set in a consistent state, with a single inline-help mechanism in production — no parallel old/new systems.

Acceptance Criteria

- Inline markers are template-declared, not hardcoded in controllers.
- Clicking a marker opens the deep-linked help page in a panel or new tab; the resolver falls back the same way as FR-17.1 (never a 404).
- Every inline marker in the production template set either links to a real help target or has been removed.
- The Opt-In Rule is documented as a project developer convention (proposed location: extension of the existing `ui-conventions` skill).
- Markers do not measurably degrade page render time.

Dependencies FR-17.1 (deep-link target, `(template_key, control_id)` naming convention), FR-17 parent (visibility and role rules apply), FR-17.3 (glossary entries are a common deep-link target from inline markers).

Notes **Wave 2 (usability).** A small backlog candidate (not part of 17.5 itself): a linter or CI check that flags non-sanctioned `title="..."` usage in templates so drift back to ad-hoc tooltips is visible at review time.

FR-17.6 — Getting Started and Onboarding

Format: user-story **Recommended Analyses:** R-F, R-C

Background New users benefit from a short guided first-run that introduces the key concepts (applications, requirements, stories, analyses) and where to find help. This is onboarding, not a feature tutorial per se.

Shall Statement As a new REQQA user, I want a short guided orientation on first login, so that I understand the main concepts and where to turn for help without reading the manual cover-to-cover.

Acceptance Criteria

- Onboarding is triggered on first login and is dismissible.
- Users can re-launch it from the help menu.
- It is tracked per user so it does not re-appear after dismissal.

Dependencies FR-17.1, FR-17.2

Notes **Wave 4 (polish/marketing).**

FR-17.7 — Release Notes and Public What's New

Format: shall-light **Recommended Analyses:** R-F, R-C

Background Release notes serve two audiences: existing users wanting to know what changed, and prospects on `reqqa.ai` wanting a feature overview. A per-entry visibility flag (`public` or `authenticated`) lets the same corpus serve both.

Shall Statement The system shall provide release-notes entries that carry a visibility flag (`public` | `authenticated`); public entries are rendered on the guest homepage as a feature / what's-new summary, authenticated entries are visible only to logged-in users.

Acceptance Criteria

- Each release-notes entry has a visibility flag.
- Guest homepage shows only `visibility=public` entries, ordered by date.
- Authenticated users see all entries.
- Editorial workflow requires the author to explicitly mark an entry public — default is authenticated.

Dependencies FR-17 parent (HR-17.P5 visibility rule).

Notes Replaces the earlier idea of a separate 'guest features list' child. **Wave 4 (polish/marketing).**

FR-17.8 — Frequently Asked Questions

Format: shall-light **Recommended Analyses:** R-F, R-C

Background A curated FAQ surface answers recurring user questions. Candidate FAQ entries are surfaced by search analytics (zero-result queries, repeated queries) and page feedback — admin reviews and publishes; auto-publishing is not permitted.

Shall Statement The system shall provide a curated FAQ surface whose entries are created only by admin review of candidates, not auto-published from logs or AI responses.

Acceptance Criteria

- FAQ entries are first-class help content, searchable via FR-17.4.
- A candidate queue lists high-volume or zero-result search terms and low-rated pages for admin triage.
- Publishing an FAQ entry requires an admin action; the source signal (search term, feedback, support question) is recorded.

Dependencies FR-17.4 (unified search), FR-17.11 (feedback), FR-17.12 (analytics).

Notes **Wave 3 (feedback loop).**

NFR-00 — Non-Functional Requirements

Priority: **Medium**

NFR-01 — Interactive Response Time Performance

Priority: **High**

The system shall respond to all interactive user actions (page loads, form submissions, navigation) within 3 seconds under normal operating conditions with up to 25 concurrent users. Database queries must be optimized with appropriate indexing to ensure sub-second response times for list views and detail pages. Background job initiation must provide immediate feedback to users within 500 milliseconds, even though the job itself may take minutes to complete. The system shall implement caching strategies for frequently accessed data such as glossary terms and catalogue listings to minimize database load and improve perceived performance.

NFR-02 — AI Operation Completion Windows

Priority: **High**

The system shall complete mission statement and story analysis operations within 30-120 seconds and requirements generation operations within 60-180 seconds under normal AI API conditions. The system must handle AI provider rate limits and timeouts gracefully, implementing exponential backoff retry logic for transient failures. When AI operations exceed expected completion windows, the system shall notify users of the delay and provide options to cancel and retry. All AI operations must preserve partial results so that failures do not result in complete loss of work, allowing users to resume or retry from the last successful step.

NFR-03 — Data Security and Encryption

Priority: **High**

The system shall enforce HTTPS for all client-server communication with TLS 1.2 or higher. AI provider API keys must be encrypted at rest using AES-256 encryption with keys managed separately from application data. User sessions must expire after 24 hours of inactivity and must be invalidated immediately upon logout. The system shall implement SQL injection prevention through parameterized queries in the PyDAL ORM layer and must sanitize all user input to prevent cross-site scripting attacks. Database backups must be encrypted and stored securely with access restricted to authorized operations personnel only.

NFR-04 — Audit Trail and Change Tracking

Priority: **Medium**

The system shall maintain a complete, immutable audit trail for all create, update, and delete operations on mission statements, requirements, stories, and issues. Each audit record must capture the user who performed the action, timestamp with timezone, the specific fields changed, and both old and new values. Audit logs must be retained for the lifetime of the organization's account and must be queryable by administrators for compliance and troubleshooting purposes. The system shall provide audit trail visualization showing the evolution of artifacts over time, allowing users to understand how requirements and stories developed through iterative refinement.

NFR-05 — Usability and Workflow Guidance

Priority: **Medium**

The system shall provide contextual guidance throughout the specification-to-quality workflow, helping users understand the recommended sequence of activities: mission creation, mission analysis, requirements generation, requirements refinement, story creation, story analysis, and issue resolution. The user interface must clearly indicate the current state of each artifact (draft, analyzed, approved) and must highlight when artifacts require attention due to open issues or outdated analysis. Error messages and validation feedback must be specific and actionable, telling users not just what is wrong but how to fix it. The system shall support keyboard navigation and must be usable on standard desktop and laptop displays with minimum resolution of 1366x768 pixels.

TR-00 — Technical Requirements

Priority: **Medium**

TR-01 — Technology Stack and Framework Constraints

Priority: **High**

The system shall be implemented using py4web framework with Python 3.8 or higher, PyDAL ORM for database abstraction, and MySQL 8.0 or higher as the primary data store. The frontend shall use Bulma CSS framework for responsive layout and styling, with vanilla JavaScript or minimal framework usage for interactive behaviors. Background job processing shall use Redis Queue (RQ) with Redis 6.0 or higher as the message broker and result backend. All new features and components must integrate with this existing technology stack without introducing additional frameworks or requiring architectural changes. The system shall follow py4web conventions for routing, controllers, and view templates.

TR-02 — AI Provider Integration Architecture

Priority: **High**

The system shall integrate with OpenAI and Anthropic APIs using organization-specific API keys configured through the administrative interface. API client implementations must support configurable timeout values, retry logic with exponential backoff, and graceful degradation when providers are unavailable. The system shall abstract AI provider interactions behind a common interface to allow switching between providers or adding new providers without changing business logic. All AI API requests and responses must be logged for debugging and cost tracking purposes, with sensitive content redacted from logs. The system shall implement rate limiting at the application level to prevent individual users or organizations from exhausting API quotas.

TR-03 — Browser and Platform Compatibility

Priority: **Medium**

The system shall support current and previous major versions of Chrome, Firefox, Safari, and Edge browsers on Windows, macOS, and Linux desktop operating systems. The application must function correctly with JavaScript enabled and must gracefully degrade or display appropriate messages when JavaScript is disabled. The system shall be responsive to viewport sizes from 1366x768 pixels upward but is not required to support mobile phone or tablet form factors at MVP stage. All user interface components must render correctly and maintain functionality across supported browsers without requiring browser-specific code or polyfills for modern web standards.

TR-04 — Data Import and Export Capabilities

Priority: **Medium**

The system shall provide import capabilities for requirements and stories from structured formats including JSON and CSV, with field mapping configuration to accommodate variations in source data structure. Export functionality must generate valid JSON conforming to a documented schema, Markdown with proper heading hierarchy and formatting, and PDF with professional styling suitable for formal documentation. The system shall validate imported data for required fields, data type correctness, and referential integrity before committing to the database, providing detailed error reports for invalid imports. Export operations must handle large datasets efficiently, supporting pagination or streaming for applications with hundreds of requirements and stories.

TR-05 — Deployment and Scaling Architecture

Priority: **Low**

The system shall be deployed as a cloud-hosted SaaS application on a single server configuration suitable for up to 25 concurrent users at MVP stage. The deployment architecture must support vertical scaling (increased CPU, memory, storage) without application code changes. Database connection pooling must be configured to handle concurrent requests efficiently, and Redis Queue workers must be configurable to process multiple background jobs in parallel. The system shall include health check endpoints for monitoring service availability and must support zero-downtime deployments through graceful shutdown of background workers and connection draining. Application logs must be structured and centralized to support operational monitoring and troubleshooting.

User stories

Stories for FR-18.3.1 — Template Recommended Analyses Metadata

Manage Recommended Analyses on Requirement Templates

```
@feature:manage-recommended-analyses-on-requirement-templat @req:FR-18.3.1 @priority:medium
Feature: Manage Recommended Analyses on Requirement Templates

  As a Template Administrator
  I want to configure which analyser step-codes are recommended by default for each requirement
    template
  So that requirements created from the template automatically inherit appropriate analysis
    workflows

  Background:
    Given the analyser step-code registry contains the following registered codes:
      | step_code | description                    |
      | R-D       | Requirements Decomposition     |
      | R-F       | Requirements Feasibility       |
      | R-C       | Requirements Completeness      |
      | R-T       | Requirements Testability       |
      | R-V       | Requirements Verification      |
    And a requirement template "Standard Functional Requirement" exists

  Rule: Template administrators can configure recommended analyses

    @happy-path @automation-ready @test:required
    Scenario: Configure valid recommended analyses on a template
      Given the template "Standard Functional Requirement" has no recommended analyses configured
      When the Template Administrator sets the recommended analyses to "R-D,R-F,R-C"
      Then the template "Standard Functional Requirement" is saved successfully
      And the template view displays the recommended analyses as "R-D, R-F, R-C" under the
    "Recommended Analyses" section

    @happy-path @automation-ready @test:required
    Scenario: Update existing recommended analyses on a template
      Given the template "Standard Functional Requirement" has recommended analyses "R-D,R-F"
      When the Template Administrator updates the recommended analyses to "R-D,R-F,R-C,R-T"
      Then the template "Standard Functional Requirement" is saved successfully
      And the template view displays the recommended analyses as "R-D, R-F, R-C, R-T" under the
    "Recommended Analyses" section

    @edge-case @empty-input @automation-ready @test:required
    Scenario: Clear all recommended analyses from a template
      Given the template "Standard Functional Requirement" has recommended analyses "R-D,R-F,R-C"
      When the Template Administrator clears the recommended analyses field
      Then the template "Standard Functional Requirement" is saved successfully
      And the template view displays "No default recommendations" under the "Recommended
    Analyses" section

    @happy-path @empty-input @automation-ready @test:required
    Scenario: Create a template with no recommended analyses
      Given a new requirement template "Minimal Template" is being created
      When the Template Administrator leaves the recommended analyses field empty
      Then the template "Minimal Template" is saved successfully
      And the template view displays "No default recommendations" under the "Recommended
    Analyses" section
```

```
Rule: BR-18.3.1.R1 -- Field is validated against registered analyser step-codes

  @error-path @invalid-input @validation @automation-ready @test:required
  Scenario: Reject template save with unknown analyser step-code
    Given the template "Standard Functional Requirement" has recommended analyses "R-D,R-F"
    When the Template Administrator attempts to set the recommended analyses to
  "R-D,R-F,R-UNKNOWN"
    Then the save operation is rejected
    And the system displays the error message "Invalid analyser step-code: R-UNKNOWN is not
  registered in the analyser step-code registry"
    And the template "Standard Functional Requirement" retains its previous recommended
  analyses "R-D,R-F"

  @error-path @invalid-input @validation @automation-ready @test:required
  Scenario: Reject template save with multiple unknown analyser step-codes
    Given the template "Standard Functional Requirement" has no recommended analyses configured
    When the Template Administrator attempts to set the recommended analyses to
  "R-D,R-INVALID,R-F,R-FAKE"
    Then the save operation is rejected
    And the system displays the error message "Invalid analyser step-codes: R-INVALID, R-FAKE
  are not registered in the analyser step-code registry"
    And the template "Standard Functional Requirement" has no recommended analyses configured

  @error-path @invalid-input @validation @automation-ready @test:required
  Scenario: Reject template save with malformed step-code format
    Given the template "Standard Functional Requirement" has recommended analyses "R-D,R-F"
    When the Template Administrator attempts to set the recommended analyses to "R-D, , R-F"
    Then the save operation is rejected
    And the system displays the error message "Invalid analyser step-code: empty or
  whitespace-only code is not allowed"
    And the template "Standard Functional Requirement" retains its previous recommended
  analyses "R-D,R-F"

Rule: BR-18.3.1.R2 -- Empty field is valid and means no default recommendation

  @edge-case @empty-input @automation-ready @test:required
  Scenario: Requirements created from template with empty recommended analyses have no default
  analyses
    Given the template "Minimal Template" has no recommended analyses configured
    When a requirement "REQ-001" is created from the template "Minimal Template"
    Then the requirement "REQ-001" has no recommended analyses inherited
    And the requirement "REQ-001" analysis kickoff UI shows no pre-selected analyses

Rule: BR-18.3.1.R3 -- Changes to template recommended analyses do not propagate to existing
  requirements

  @edge-case @data-integrity @automation-ready @test:required
  Scenario: Updating template recommended analyses does not affect existing requirements
    Given the template "Standard Functional Requirement" has recommended analyses "R-D,R-F"
    And a requirement "REQ-100" was created from the template "Standard Functional
  Requirement" with inherited analyses "R-D,R-F"
    When the Template Administrator updates the template recommended analyses to
  "R-D,R-F,R-C,R-T"
    Then the template "Standard Functional Requirement" displays recommended analyses "R-D,
  R-F, R-C, R-T"
    But the requirement "REQ-100" retains its recommended analyses as "R-D,R-F"
    And the requirement "REQ-100" is not automatically updated

  @edge-case @data-integrity @automation-ready @test:required
```

```
  Scenario: Clearing template recommended analyses does not affect existing requirements
    Given the template "Standard Functional Requirement" has recommended analyses "R-D,R-F,R-C"
    And a requirement "REQ-200" was created from the template "Standard Functional
  Requirement" with inherited analyses "R-D,R-F,R-C"
    When the Template Administrator clears the template recommended analyses
    Then the template "Standard Functional Requirement" displays "No default recommendations"
    But the requirement "REQ-200" retains its recommended analyses as "R-D,R-F,R-C"
    And the requirement "REQ-200" is not automatically updated

Rule: Template view renders recommended analyses visibly

  @happy-path @automation-ready @test:required
  Scenario: Template view displays configured recommended analyses
    Given the template "Standard Functional Requirement" has recommended analyses "R-D,R-F,R-C"
    When the Template Administrator views the template "Standard Functional Requirement"
    Then the template view displays a section labelled "Recommended Analyses"
    And the section contains the analyser step-codes "R-D, R-F, R-C"
    And each step-code is rendered in a human-readable format

  @edge-case @empty-input @automation-ready @test:required
  Scenario: Template view displays empty state for templates with no recommended analyses
    Given the template "Minimal Template" has no recommended analyses configured
    When the Template Administrator views the template "Minimal Template"
    Then the template view displays a section labelled "Recommended Analyses"
    And the section contains the text "No default recommendations"

Rule: Template edit UI provides controls for managing recommended analyses

  @happy-path @automation-ready @test:required
  Scenario: Template edit UI allows selection of multiple recommended analyses
    Given the Template Administrator is editing the template "Standard Functional Requirement"
    And the template currently has recommended analyses "R-D,R-F"
    When the Template Administrator selects the additional analyses "R-C" and "R-T" using the
  checkbox control
    And the Template Administrator saves the template
    Then the template "Standard Functional Requirement" has recommended analyses
  "R-D,R-F,R-C,R-T"

  @happy-path @automation-ready @test:required
  Scenario: Template edit UI allows deselection of recommended analyses
    Given the Template Administrator is editing the template "Standard Functional Requirement"
    And the template currently has recommended analyses "R-D,R-F,R-C,R-T"
    When the Template Administrator deselects the analyses "R-C" and "R-T" using the checkbox
  control
    And the Template Administrator saves the template
    Then the template "Standard Functional Requirement" has recommended analyses "R-D,R-F"

Rule: Data schema supports recommended analyses storage

  @data-integrity @automation-ready @test:required
  Scenario: Recommended analyses are persisted in the database
    Given the template "Standard Functional Requirement" has no recommended analyses configured
    When the Template Administrator sets the recommended analyses to "R-D,R-F,R-C"
    Then the database column "req_templates.recommended_analyses" for template "Standard
  Functional Requirement" contains "R-D,R-F,R-C"

  @boundary @max-value @automation-ready @test:required
  Scenario: Store maximum length recommended analyses string
    Given the template "Comprehensive Template" has no recommended analyses configured
```

```
    When the Template Administrator sets the recommended analyses to a comma-separated list of
  50 valid step-codes
    Then the template "Comprehensive Template" is saved successfully
    And all 50 step-codes are stored in the "req_templates.recommended_analyses" column

  @error-path @max-value @validation @automation-ready @test:required
  Scenario: Reject recommended analyses exceeding maximum field length
    Given the template "Comprehensive Template" has no recommended analyses configured
    When the Template Administrator attempts to set the recommended analyses to a
  comma-separated list exceeding 255 characters
    Then the save operation is rejected
    And the system displays the error message "Recommended analyses field exceeds maximum
  length of 255 characters"
    And the template "Comprehensive Template" has no recommended analyses configured

Rule: Integration -- Analysis kickoff UI consumes template recommended analyses (FR-18.3.3)

  @integration @kickoff-ui @automation-ready @test:required
  Scenario: Kickoff UI pre-ticks analyses inherited from the template via the requirement
    Given the template "Standard Functional Requirement" has recommended analyses "R-D,R-F,R-C"
    And a requirement "REQ-300" was created from the template "Standard Functional
  Requirement" inheriting "R-D,R-F,R-C"
    When the analyst opens the analysis kickoff UI for requirement "REQ-300"
    Then the kickoff UI checkbox list pre-ticks the analyses "R-D, R-F, R-C"
    And other registered analyses are shown unticked
    And the analyst can adjust the selection before running the analysis

  @integration @kickoff-ui @edge-case @automation-ready @test:required
  Scenario: Kickoff UI shows nothing pre-selected when template has no recommended analyses
    Given the template "Minimal Template" has no recommended analyses configured
    And a requirement "REQ-301" was created from the template "Minimal Template" with no
  inherited analyses
    When the analyst opens the analysis kickoff UI for requirement "REQ-301"
    Then the kickoff UI checkbox list shows all registered analyses unticked
    And the analyst must select analyses manually before running

  @integration @kickoff-ui @edge-case @automation-ready @test:required
  Scenario: Kickoff UI ignores recommended step-codes that are no longer registered
    Given a requirement "REQ-302" has recommended analyses "R-D,R-OBSOLETE,R-F"
    And the analyser step-code registry does not contain "R-OBSOLETE"
    When the analyst opens the analysis kickoff UI for requirement "REQ-302"
    Then the kickoff UI pre-ticks "R-D, R-F" only
    And the kickoff UI displays a non-blocking notice "1 unrecognised step-code in recommended
  analyses: R-OBSOLETE"
```

Stories for FR-18.3.2 — Requirement Recommended Analyses

Persist Recommended Analyses in Database Schema

```
@feature:persist-recommended-analyses-in-database-schema @req:FR-18.3.2 @priority:medium @
    data-integrity @test:required
Feature: Persist Recommended Analyses in Database Schema

  As a System
  I want to store recommended_analyses in both the requirements and requirements_history tables
  So that analysis recommendations are versioned and auditable

  Background:
    Given the database schema includes the "requirements" table
```

```
  And the database schema includes the "requirements_history" table
  And the analyser step-code registry contains "R-D, R-F, R-C, R-V, R-T"

@happy-path @automation-ready @test:required
Scenario: Column exists in requirements table with correct specification
  When the system inspects the "requirements" table schema
  Then a column named "recommended_analyses" exists
  And the column data type is "VARCHAR(255)"
  And the column allows NULL values

@happy-path @automation-ready @test:required
Scenario: Column exists in requirements_history table with correct specification
  When the system inspects the "requirements_history" table schema
  Then a column named "recommended_analyses" exists
  And the column data type is "VARCHAR(255)"
  And the column allows NULL values

@happy-path @automation-ready @test:required
Scenario: Store comma-separated step-codes in requirements table
  Given a requirement with ID "REQ-001" exists
  When the system stores "R-D,R-F,R-C" in the recommended_analyses column for requirement
  "REQ-001"
  Then the requirements table contains "R-D,R-F,R-C" in the recommended_analyses column for
  requirement "REQ-001"

@happy-path @automation-ready @test:required
Scenario: Store comma-separated step-codes in requirements_history table
  Given a requirement with ID "REQ-002" exists
  And the requirement has been modified 3 times
  When the system stores "R-V,R-T" in the recommended_analyses column for the latest history
  entry of requirement "REQ-002"
  Then the requirements_history table contains "R-V,R-T" in the recommended_analyses column
  for the latest history entry of requirement "REQ-002"

@happy-path @edge-case @empty-input @automation-ready @test:required
Scenario: Store empty recommended_analyses value
  Given a requirement with ID "REQ-003" exists
  When the system stores an empty value in the recommended_analyses column for requirement
  "REQ-003"
  Then the requirements table contains NULL in the recommended_analyses column for requirement
  "REQ-003"

@happy-path @edge-case @automation-ready @test:required
Scenario: Store single step-code without comma
  Given a requirement with ID "REQ-004" exists
  When the system stores "R-D" in the recommended_analyses column for requirement "REQ-004"
  Then the requirements table contains "R-D" in the recommended_analyses column for
  requirement "REQ-004"

@happy-path @data-integrity @automation-ready @test:required
Scenario: Version history tracks changes to recommended_analyses
  Given a requirement with ID "REQ-005" exists
  And the requirement has recommended_analyses "R-D,R-F"
  When the requirement is updated with recommended_analyses "R-D,R-F,R-C"
  Then the requirements_history table contains a record with recommended_analyses "R-D,R-F"
  for requirement "REQ-005"
  And the requirements_history table contains a record with recommended_analyses "R-D,R-F,R-C"
  for requirement "REQ-005"
  And the requirements table contains "R-D,R-F,R-C" in the recommended_analyses column for
```

```
  requirement "REQ-005"

@happy-path @data-integrity @automation-ready @test:required
Scenario: Preserve recommended_analyses through multiple requirement updates
  Given a requirement with ID "REQ-006" exists
  And the requirement has recommended_analyses "R-D"
  When the requirement title is updated to "Updated Requirement Title"
  And the requirement description is updated to "Updated description text"
  Then the requirements table contains "R-D" in the recommended_analyses column for
  requirement "REQ-006"
  And the requirements_history table contains 3 records for requirement "REQ-006"
  And all history records for requirement "REQ-006" contain "R-D" in the recommended_analyses
  column

@boundary @max-value @automation-ready @test:required
Scenario: Store maximum length comma-separated list
  Given a requirement with ID "REQ-007" exists
  And a comma-separated list of step-codes with total length 255 characters
  When the system stores the 255-character list in the recommended_analyses column for
  requirement "REQ-007"
  Then the requirements table contains the complete 255-character list for requirement
  "REQ-007"

@error-path @invalid-input @validation @automation-ready @test:required
Scenario: Reject step-codes exceeding column length
  Given a requirement with ID "REQ-008" exists
  And a comma-separated list of step-codes with total length 256 characters
  When the system attempts to store the 256-character list in the recommended_analyses column
  for requirement "REQ-008"
  Then the system rejects the value with a data truncation error
  And the requirements table does not contain the 256-character list for requirement "REQ-008"

@happy-path @data-integrity @automation-ready @test:required
Scenario: Maintain referential integrity between requirements and requirements_history
  Given a requirement with ID "REQ-009" exists
  And the requirement has recommended_analyses "R-F,R-C"
  When the requirement is updated 5 times with different recommended_analyses values
  Then the requirements_history table contains 6 records for requirement "REQ-009"
  And each history record for requirement "REQ-009" has a non-null recommended_analyses value
  And the latest history record matches the current requirements table value

@edge-case @data-integrity @automation-ready @test:required
Scenario: Handle NULL to non-NULL transition
  Given a requirement with ID "REQ-010" exists
  And the requirement has NULL recommended_analyses
  When the system stores "R-D,R-F" in the recommended_analyses column for requirement "REQ-010"
  Then the requirements table contains "R-D,R-F" in the recommended_analyses column for
  requirement "REQ-010"
  And the requirements_history table contains a record with NULL recommended_analyses for
  requirement "REQ-010"
  And the requirements_history table contains a record with "R-D,R-F" recommended_analyses for
  requirement "REQ-010"

@edge-case @data-integrity @automation-ready @test:required
Scenario: Handle non-NULL to NULL transition
  Given a requirement with ID "REQ-011" exists
  And the requirement has recommended_analyses "R-V,R-T"
  When the system stores NULL in the recommended_analyses column for requirement "REQ-011"
  Then the requirements table contains NULL in the recommended_analyses column for requirement
```

```
    "REQ-011"
    And the requirements_history table contains a record with "R-V,R-T" recommended_analyses for
    requirement "REQ-011"
    And the requirements_history table contains a record with NULL recommended_analyses for
    requirement "REQ-011"

  @happy-path @data-integrity @automation-ready @test:required
  Scenario: Store step-codes matching analyses.stepcode format
    Given a requirement with ID "REQ-012" exists
    And the analyses table contains records with stepcode values "R-D, R-F, R-C"
    When the system stores "R-D,R-F,R-C" in the recommended_analyses column for requirement
    "REQ-012"
    Then the recommended_analyses format matches the analyses.stepcode format
    And the requirements table contains "R-D,R-F,R-C" for requirement "REQ-012"

  Rule: Integration -- Schema column is populated by the template inheritance flow (FR-18.3.1)

    @integration @template-system @automation-ready @test:required
    Scenario: Schema accepts recommended_analyses written by the template-seeding flow
      Given a requirement is being created from a template carrying recommended_analyses
    "R-D,R-F,R-C"
      When the requirement is persisted by the template-seeding flow
      Then the requirements.recommended_analyses column for that requirement contains
    "R-D,R-F,R-C"
      And the value is retrievable via the standard requirement read API

    @integration @template-system @edge-case @automation-ready @test:required
    Scenario: Schema accepts NULL/empty recommended_analyses on requirements without a template
      Given a requirement is being created without a template
      When the requirement is persisted
      Then the requirements.recommended_analyses column is NULL or empty for that requirement

  Rule: Integration -- Recommended analyses are read by the analysis execution workflow
    (FR-18.3.3)

    @integration @analysis-execution @automation-ready @test:required
    Scenario: Analysis execution reads recommended_analyses to determine which analysers to run
      Given a requirement has recommended_analyses "R-D,R-F"
      When the analysis kickoff UI is launched for that requirement
      Then the kickoff UI fetches the recommended_analyses value from the schema column
      And uses it to pre-tick the analyser checkbox list (per FR-18.3.3)

    @integration @analysis-execution @audit @automation-ready @test:required
    Scenario: Analysis run records the step-codes actually selected (not the recommendation) on
    the analyses row
      Given a requirement has recommended_analyses "R-D,R-F"
      And the analyst overrides the selection to "R-D,R-F,R-C" at kickoff
      When the analysis is run
      Then the analyses.stepCode column for that run contains "R-D,R-F,R-C"
      And the requirement's recommended_analyses field is unchanged
```

Preserve Recommended Analyses on Template Removal

```
@feature:preserve-recommended-analyses-on-template-removal @req:FR-18.3.2 @priority:medium
Feature: Preserve Recommended Analyses on Template Removal

  As a Requirement Author
  I want my custom recommended_analyses to be preserved when I remove a template from a
    requirement
```

```
So that I do not lose my analysis configuration

Background:
  Given the requirement "REQ-001" exists with template "Template-A"
  And the requirement "REQ-001" has recommended_analyses "R-D,R-F,R-C"

@happy-path @test:required @automation-ready @data-integrity
Scenario: Template removal preserves existing recommended analyses
  When the Requirement Author removes the template from requirement "REQ-001"
  Then the requirement "REQ-001" has no template assigned
  And the requirement "REQ-001" still has recommended_analyses "R-D,R-F,R-C"

@happy-path @test:required @automation-ready @data-integrity
Scenario: Template removal preserves custom recommended analyses different from template
   default
  Given the requirement "REQ-002" exists with template "Template-B"
  And template "Template-B" has recommended_analyses "R-D,R-F"
  And the Requirement Author has overridden recommended_analyses to "R-C,R-V" for requirement
  "REQ-002"
  When the Requirement Author removes the template from requirement "REQ-002"
  Then the requirement "REQ-002" has no template assigned
  And the requirement "REQ-002" still has recommended_analyses "R-C,R-V"

@edge-case @test:required @automation-ready @empty-input @data-integrity
Scenario: Template removal preserves empty recommended analyses
  Given the requirement "REQ-003" exists with template "Template-C"
  And the Requirement Author has cleared recommended_analyses for requirement "REQ-003"
  When the Requirement Author removes the template from requirement "REQ-003"
  Then the requirement "REQ-003" has no template assigned
  And the requirement "REQ-003" has empty recommended_analyses

@happy-path @test:required @automation-ready @data-integrity
Scenario: Requirement continues to function independently after template removal
  Given the requirement "REQ-004" exists with template "Template-D"
  And the requirement "REQ-004" has recommended_analyses "R-D,R-F,R-C"
  When the Requirement Author removes the template from requirement "REQ-004"
  And the Requirement Author edits recommended_analyses to "R-D,R-V" for requirement "REQ-004"
  Then the requirement "REQ-004" has recommended_analyses "R-D,R-V"
  And the requirement "REQ-004" has no template assigned

@edge-case @test:required @automation-ready @data-integrity
Scenario: Template removal does not affect other requirements using the same template
  Given the requirement "REQ-005" exists with template "Template-E"
  And the requirement "REQ-006" exists with template "Template-E"
  And both requirements have recommended_analyses "R-D,R-F"
  When the Requirement Author removes the template from requirement "REQ-005"
  Then the requirement "REQ-005" has no template assigned
  And the requirement "REQ-005" still has recommended_analyses "R-D,R-F"
  And the requirement "REQ-006" still has template "Template-E"
  And the requirement "REQ-006" still has recommended_analyses "R-D,R-F"

@happy-path @test:required @automation-ready @data-integrity @audit-trail
Scenario: Template removal preserves recommended analyses in version history
  Given the requirement "REQ-007" exists with template "Template-F"
  And the requirement "REQ-007" has recommended_analyses "R-D,R-F,R-C"
  When the Requirement Author removes the template from requirement "REQ-007"
  Then the requirements_history table contains a record for requirement "REQ-007"
  And the history record shows recommended_analyses "R-D,R-F,R-C"
  And the history record shows template removal
```

```
@alternate-path @test:required @automation-ready @data-integrity
Scenario: Template removal followed by template reapplication offers merge options
  Given the requirement "REQ-008" exists with template "Template-G"
  And the requirement "REQ-008" has recommended_analyses "R-D,R-F"
  When the Requirement Author removes the template from requirement "REQ-008"
  And the Requirement Author applies template "Template-H" to requirement "REQ-008"
  And template "Template-H" has recommended_analyses "R-C,R-V"
  Then the system prompts the Requirement Author with options: Replace, Keep, Merge
  And the default option is Keep

@edge-case @test:required @automation-ready @data-integrity
Scenario: Multiple template removals preserve recommended analyses consistently
  Given the requirement "REQ-009" exists with template "Template-I"
  And the requirement "REQ-009" has recommended_analyses "R-D,R-F,R-C"
  When the Requirement Author removes the template from requirement "REQ-009"
  And the Requirement Author applies template "Template-J" to requirement "REQ-009"
  And the Requirement Author chooses Keep to retain recommended_analyses "R-D,R-F,R-C"
  And the Requirement Author removes the template from requirement "REQ-009" again
  Then the requirement "REQ-009" has no template assigned
  And the requirement "REQ-009" still has recommended_analyses "R-D,R-F,R-C"

Rule: User flow -- How a template is removed from a requirement

  @happy-path @template-removal @automation-ready @test:required
  Scenario: Requirement Author removes the template from a requirement via the requirement
  edit form
    Given a requirement "REQ-500" was created from the template "Functional Requirement"
  inheriting recommended_analyses "R-D,R-F,R-C"
    When the Requirement Author opens the requirement edit form for "REQ-500"
    And clears the Template field (sets it to none / empty)
    And saves the requirement
    Then the requirement "REQ-500" no longer has an associated template
    And the requirement "REQ-500" retains its recommended_analyses "R-D,R-F,R-C"

  @alternate-path @template-removal @automation-ready @test:required
  Scenario: Bulk template removal across multiple requirements (administrator action -- out of
  scope for v1)
    Given an administrator wishes to remove a deprecated template from all requirements that
  reference it
    When no bulk-removal mechanism exists in v1
    Then the administrator must remove the template per requirement using the requirement edit
  form
    And future bulk-removal tooling is captured as a backlog item, not part of FR-18.3.x v1

Rule: Integration -- Preserved recommended_analyses are honoured by the analysis engine
  (FR-18.3.3)

  @integration @analysis-engine @automation-ready @test:required
  Scenario: Analysis engine reads the preserved recommended_analyses on a requirement after
  template removal
    Given a requirement "REQ-501" originally inherited recommended_analyses "R-D,R-F,R-C" from
  a template
    And the template has been removed from the requirement
    And the requirement retained its recommended_analyses "R-D,R-F,R-C"
    When the analyst opens the analysis kickoff UI for requirement "REQ-501"
    Then the kickoff UI pre-ticks the analyses "R-D, R-F, R-C"
    And the analysis engine treats the recommended_analyses identically whether seeded or
  retained
```

Prompt User on Template Change with Merge Options

```
@feature:prompt-user-on-template-change-with-merge-options
@req:FR-18.3.2
@priority:medium
@test-characteristic:data-integrity
@risk:data-loss
Feature: Prompt User on Template Change with Merge Options

  As a Requirement Author
  I want to be prompted with clear options when I change a requirement's template
  So that I can decide whether to keep my custom analysis recommendations or adopt the new
    template's defaults

  Background:
    Given the analyser step-code registry contains the following codes:
      | step_code |
      | R-D       |
      | R-F       |
      | R-C       |
      | R-V       |
      | R-T       |
    And a requirement template "Safety Requirements" exists with recommended_analyses
    "R-D,R-F,R-C"
    And a requirement template "Performance Requirements" exists with recommended_analyses
    "R-V,R-T"

  @happy-path
  @test:required
  @automation-ready
  Scenario: Keep existing recommended analyses when template is changed
    Given a requirement "REQ-001" exists with template "Safety Requirements"
    And the requirement has recommended_analyses "R-D,R-F,R-C"
    And the Requirement Author has customised the recommended_analyses to "R-D,R-V"
    When the Requirement Author changes the template to "Performance Requirements"
    Then the system presents a prompt with three options: "Replace", "Keep", "Merge"
    And the default option is "Keep"
    When the Requirement Author selects "Keep"
    Then the requirement's recommended_analyses remains "R-D,R-V"
    And the requirement's template is "Performance Requirements"

  @happy-path
  @test:required
  @automation-ready
  Scenario: Replace existing recommended analyses with new template defaults
    Given a requirement "REQ-002" exists with template "Safety Requirements"
    And the requirement has recommended_analyses "R-D,R-F,R-C"
    And the Requirement Author has customised the recommended_analyses to "R-D,R-V"
    When the Requirement Author changes the template to "Performance Requirements"
    And the system presents the merge options prompt
    And the Requirement Author selects "Replace"
    Then the requirement's recommended_analyses is overwritten to "R-V,R-T"
    And the requirement's template is "Performance Requirements"

  @happy-path
  @test:required
  @automation-ready
  Scenario: Merge existing and new template recommended analyses
    Given a requirement "REQ-003" exists with template "Safety Requirements"
```

```
  And the requirement has recommended_analyses "R-D,R-F,R-C"
  And the Requirement Author has customised the recommended_analyses to "R-D,R-V"
  When the Requirement Author changes the template to "Performance Requirements"
  And the system presents the merge options prompt
  And the Requirement Author selects "Merge"
  Then the requirement's recommended_analyses is set to the union "R-D,R-V,R-T"
  And the requirement's template is "Performance Requirements"

@edge-case
@test:required
@automation-ready
@empty-input
Scenario: Keep option preserves empty recommended analyses
  Given a requirement "REQ-004" exists with template "Safety Requirements"
  And the requirement has recommended_analyses ""
  When the Requirement Author changes the template to "Performance Requirements"
  And the system presents the merge options prompt
  And the Requirement Author selects "Keep"
  Then the requirement's recommended_analyses remains ""
  And the requirement's template is "Performance Requirements"

@edge-case
@test:required
@automation-ready
@empty-input
Scenario: Replace option overwrites empty recommended analyses with template defaults
  Given a requirement "REQ-005" exists with template "Safety Requirements"
  And the requirement has recommended_analyses ""
  When the Requirement Author changes the template to "Performance Requirements"
  And the system presents the merge options prompt
  And the Requirement Author selects "Replace"
  Then the requirement's recommended_analyses is set to "R-V,R-T"
  And the requirement's template is "Performance Requirements"

@edge-case
@test:required
@automation-ready
@empty-input
Scenario: Merge option with empty current recommended analyses adopts template defaults
  Given a requirement "REQ-006" exists with template "Safety Requirements"
  And the requirement has recommended_analyses ""
  When the Requirement Author changes the template to "Performance Requirements"
  And the system presents the merge options prompt
  And the Requirement Author selects "Merge"
  Then the requirement's recommended_analyses is set to "R-V,R-T"
  And the requirement's template is "Performance Requirements"

@edge-case
@test:required
@automation-ready
Scenario: Merge eliminates duplicate step-codes
  Given a requirement "REQ-007" exists with template "Safety Requirements"
  And the requirement has recommended_analyses "R-D,R-V"
  And a requirement template "Hybrid Requirements" exists with recommended_analyses
  "R-V,R-T,R-D"
  When the Requirement Author changes the template to "Hybrid Requirements"
  And the system presents the merge options prompt
  And the Requirement Author selects "Merge"
  Then the requirement's recommended_analyses is set to "R-D,R-V,R-T"
```

```
  And each step-code appears exactly once

@error-path
@test:required
@automation-ready
Scenario: Prompt is not shown when current recommended analyses match new template
  Given a requirement "REQ-008" exists with template "Safety Requirements"
  And the requirement has recommended_analyses "R-D,R-F,R-C"
  And a requirement template "Identical Requirements" exists with recommended_analyses
  "R-D,R-F,R-C"
  When the Requirement Author changes the template to "Identical Requirements"
  Then the system does not present the merge options prompt
  And the requirement's recommended_analyses remains "R-D,R-F,R-C"
  And the requirement's template is "Identical Requirements"

@alternate-path
@test:required
@automation-ready
Scenario: Default Keep option is applied when prompt is dismissed
  Given a requirement "REQ-009" exists with template "Safety Requirements"
  And the requirement has recommended_analyses "R-D,R-V"
  When the Requirement Author changes the template to "Performance Requirements"
  And the system presents the merge options prompt
  And the Requirement Author dismisses the prompt without selecting an option
  Then the requirement's recommended_analyses remains "R-D,R-V"
  And the requirement's template is "Performance Requirements"

@edge-case
@test:required
@automation-ready
Scenario: Template change to template with empty recommended analyses
  Given a requirement "REQ-010" exists with template "Safety Requirements"
  And the requirement has recommended_analyses "R-D,R-F,R-C"
  And a requirement template "Minimal Requirements" exists with recommended_analyses ""
  When the Requirement Author changes the template to "Minimal Requirements"
  And the system presents the merge options prompt
  And the Requirement Author selects "Replace"
  Then the requirement's recommended_analyses is set to ""
  And the requirement's template is "Minimal Requirements"

@test-characteristic:data-integrity
@test:required
@automation-ready
Scenario: Recommended analyses are preserved in requirements_history on template change
  Given a requirement "REQ-011" exists with template "Safety Requirements"
  And the requirement has recommended_analyses "R-D,R-V"
  And the requirement has 3 historical versions in requirements_history
  When the Requirement Author changes the template to "Performance Requirements"
  And the system presents the merge options prompt
  And the Requirement Author selects "Keep"
  Then a new version is recorded in requirements_history
  And the historical version contains recommended_analyses "R-D,R-V"
  And the historical version contains template "Performance Requirements"

@edge-case
@test:required
@automation-ready
Scenario: Prompt is shown when changing from no template to a template with recommended
  analyses
```

```
    Given a requirement "REQ-012" exists with no template
    And the requirement has recommended_analyses "R-D,R-V"
    When the Requirement Author assigns the template "Performance Requirements"
    Then the system presents the merge options prompt
    And the default option is "Keep"
    When the Requirement Author selects "Merge"
    Then the requirement's recommended_analyses is set to "R-D,R-V,R-T"

  @error-path
  @test:required
  @automation-ready
  Scenario: Prompt is not shown when changing from template to no template
    Given a requirement "REQ-013" exists with template "Safety Requirements"
    And the requirement has recommended_analyses "R-D,R-F,R-C"
    When the Requirement Author removes the template
    Then the system does not present the merge options prompt
    And the requirement's recommended_analyses remains "R-D,R-F,R-C"
    And the requirement has no template assigned

  Rule: Validation -- Merged step-codes are validated against the analyser registry

    @validation @merge-result @automation-ready @test:required
    Scenario: Merge filters out step-codes that are no longer registered
      Given a requirement has recommended_analyses "R-D,R-LEGACY"
      And the new template carries recommended_analyses "R-LEGACY,R-F,R-C"
      And the analyser step-code registry does not contain "R-LEGACY"
      When the user selects the "Merge" option on template change
      Then the merged recommended_analyses for the requirement is "R-D, R-F, R-C"
      And the system displays a non-blocking notice "1 unrecognised step-code dropped during
    merge: R-LEGACY"
      And the dropped step-code is logged in the requirement's audit trail

    @validation @merge-result @edge-case @automation-ready @test:required
    Scenario: Merge produces empty result when neither side carries a registered step-code
      Given a requirement has recommended_analyses "R-LEGACY-A"
      And the new template carries recommended_analyses "R-LEGACY-B"
      And neither step-code is registered
      When the user selects "Merge"
      Then the merged recommended_analyses is empty
      And the system displays a non-blocking notice naming both dropped step-codes
```

Render Recommended Analyses on Requirement View

```
@feature:render-recommended-analyses-on-requirement-view @req:FR-18.3.2 @priority:medium
Feature: Render Recommended Analyses on Requirement View

  As a Requirement Author or Stakeholder
  I want to see the recommended_analyses for a requirement displayed on the requirement view page
  So that I understand which analyses are suggested for this requirement

  Background:
    Given the system has registered analyser step-codes "R-D", "R-F", "R-C", "R-V", "R-T"
    And a requirement template "Functional Requirement" exists with recommended_analyses
    "R-D,R-F,R-C"

  @happy-path @automation-ready @test:required
  Scenario: Display recommended analyses for requirement created from template
    Given a requirement "FR-001" was created using template "Functional Requirement"
    And the requirement has recommended_analyses "R-D,R-F,R-C"
```

```
  When the Requirement Author views requirement "FR-001"
  Then the requirement view displays a "Recommended Analyses" heading
  And the recommended analyses list shows "R-D, R-F, R-C"

@happy-path @automation-ready @test:required
Scenario: Display custom recommended analyses after author override
  Given a requirement "FR-002" exists with recommended_analyses "R-V,R-T"
  When the Requirement Author views requirement "FR-002"
  Then the requirement view displays a "Recommended Analyses" heading
  And the recommended analyses list shows "R-V, R-T"

@edge-case @empty-input @automation-ready @test:required
Scenario: Display empty recommended analyses for requirement without template
  Given a requirement "FR-003" was created without a template
  And the requirement has empty recommended_analyses
  When the Requirement Author views requirement "FR-003"
  Then the requirement view displays a "Recommended Analyses" heading
  And the recommended analyses list shows no step-codes

@edge-case @automation-ready @test:required
Scenario: Display single recommended analysis
  Given a requirement "FR-004" exists with recommended_analyses "R-D"
  When the Requirement Author views requirement "FR-004"
  Then the requirement view displays a "Recommended Analyses" heading
  And the recommended analyses list shows "R-D"

@happy-path @automation-ready @test:required
Scenario: Display recommended analyses in human-readable format
  Given a requirement "FR-005" exists with recommended_analyses "R-D,R-F,R-C,R-V,R-T"
  When the Requirement Author views requirement "FR-005"
  Then the requirement view displays a "Recommended Analyses" heading
  And the recommended analyses are displayed as comma-separated step-codes
  And each step-code is clearly readable

@edge-case @automation-ready @test:required
Scenario: Display recommended analyses for retro-placeholder requirement
  Given a requirement "FR-006" exists with origin "retro-placeholder"
  And the requirement has empty recommended_analyses
  When the Requirement Author views requirement "FR-006"
  Then the requirement view displays a "Recommended Analyses" heading
  And the recommended analyses list shows no step-codes

# Note: the rendering contract is independent of how recommended_analyses got there.
# The scenarios below verify that rendering is invariant across origin paths
# (template inheritance, Keep on template change, post-removal preservation).
@happy-path @rendering-invariance @automation-ready @test:required
Scenario: Display recommended analyses after template change with Keep option
  Given a requirement "FR-007" was created using template "Functional Requirement"
  And the requirement has recommended_analyses "R-D,R-F,R-C"
  And the requirement template was changed with "Keep" option selected
  When the Requirement Author views requirement "FR-007"
  Then the requirement view displays a "Recommended Analyses" heading
  And the recommended analyses list shows "R-D, R-F, R-C"

@alternate-path @automation-ready @test:required
Scenario: Display recommended analyses after template change with Replace option
  Given a requirement "FR-008" was created using template "Functional Requirement"
  And the requirement originally had recommended_analyses "R-D,R-F,R-C"
  And a requirement template "Non-Functional Requirement" exists with recommended_analyses
```

```
    "R-V,R-T"
    And the requirement template was changed to "Non-Functional Requirement" with "Replace"
    option selected
    When the Requirement Author views requirement "FR-008"
    Then the requirement view displays a "Recommended Analyses" heading
    And the recommended analyses list shows "R-V, R-T"

  @alternate-path @automation-ready @test:required
  Scenario: Display recommended analyses after template change with Merge option
    Given a requirement "FR-009" was created using template "Functional Requirement"
    And the requirement originally had recommended_analyses "R-D,R-F"
    And a requirement template "Extended Requirement" exists with recommended_analyses "R-C,R-V"
    And the requirement template was changed to "Extended Requirement" with "Merge" option
    selected
    When the Requirement Author views requirement "FR-009"
    Then the requirement view displays a "Recommended Analyses" heading
    And the recommended analyses list shows "R-D, R-F, R-C, R-V"

  @happy-path @automation-ready @test:required
  Scenario: Display recommended analyses after template removal
    Given a requirement "FR-010" was created using template "Functional Requirement"
    And the requirement has recommended_analyses "R-D,R-F,R-C"
    And the template was removed from the requirement
    When the Requirement Author views requirement "FR-010"
    Then the requirement view displays a "Recommended Analyses" heading
    And the recommended analyses list shows "R-D, R-F, R-C"

  Rule: Integration -- Display reflects analyser registry state (FR-18.3.1, FR-18.3.3)

    @integration @analyser-registry @automation-ready @test:required
    Scenario: Display labels each step-code with its registry entry
      Given the requirement "FR-100" has recommended_analyses "R-D,R-F"
      And the analyser registry labels "R-D" as "Definitions" and "R-F" as "Features"
      When the Requirement Author views requirement "FR-100"
      Then the recommended-analyses section shows the codes "R-D, R-F"
      And each code is rendered with its registry label adjacent or on hover

    @integration @analyser-registry @edge-case @automation-ready @test:required
    Scenario: Display flags step-codes not present in the analyser registry
      Given the requirement "FR-101" has recommended_analyses "R-D,R-OBSOLETE"
      And "R-OBSOLETE" is not present in the analyser registry
      When the Requirement Author views requirement "FR-101"
      Then the recommended-analyses section shows "R-D, R-OBSOLETE"
      And "R-OBSOLETE" is visually flagged as unrecognised (e.g. struck through or with a
    warning icon)
      And the view does not error out
```

Seed Recommended Analyses from Template on Requirement Creation

```
@feature:seed-recommended-analyses-from-template-on-require @req:FR-18.3.2 @priority:medium
Feature: Seed Recommended Analyses from Template on Requirement Creation

  As a Requirement Author
  I want the recommended_analyses field to be automatically populated from the selected template
    when I create a new requirement
  So that I inherit sensible analysis defaults without manual re-entry

  Background:
    Given the analyser step-code registry contains the following codes:
```

```
    | step_code |
    | R-D       |
    | R-F       |
    | R-C       |
    | R-T       |
    | R-V       |
  And a requirement template "Standard Functional" exists with recommended_analyses
  "R-D,R-F,R-C"
  And a requirement template "Security Focused" exists with recommended_analyses "R-D,R-F,R-V"

@happy-path @automation-ready @test:required
Scenario: Requirement inherits recommended analyses from selected template on creation
  Given the Requirement Author is creating a new requirement
  And the author selects the template "Standard Functional"
  When the author saves the requirement
  Then the requirement's recommended_analyses field contains "R-D,R-F,R-C"
  And the requirement is persisted to the database

@happy-path @automation-ready @test:required
Scenario: Author edits inherited recommended analyses before saving
  Given the Requirement Author is creating a new requirement
  And the author selects the template "Standard Functional"
  And the recommended_analyses field is pre-populated with "R-D,R-F,R-C"
  When the author changes the recommended_analyses field to "R-D,R-T"
  And the author saves the requirement
  Then the requirement's recommended_analyses field contains "R-D,R-T"
  And the requirement is persisted to the database

@happy-path @automation-ready @test:required
Scenario: Author edits inherited recommended analyses after saving
  Given a requirement exists with recommended_analyses "R-D,R-F,R-C" inherited from template
  "Standard Functional"
  When the Requirement Author edits the requirement
  And the author changes the recommended_analyses field to "R-D,R-V"
  And the author saves the requirement
  Then the requirement's recommended_analyses field contains "R-D,R-V"
  And the requirement is persisted to the database

@edge-case @empty-input @automation-ready @test:required
Scenario: Requirement created without template has empty recommended analyses
  Given the Requirement Author is creating a new requirement
  And the author does not select a template
  When the author saves the requirement
  Then the requirement's recommended_analyses field is empty
  And the requirement is persisted to the database

@alternate-path @automation-ready @test:required
Scenario: Requirement inherits different analyses from different template
  Given the Requirement Author is creating a new requirement
  And the author selects the template "Security Focused"
  When the author saves the requirement
  Then the requirement's recommended_analyses field contains "R-D,R-F,R-V"
  And the requirement is persisted to the database

@edge-case @empty-input @automation-ready @test:required
Scenario: Requirement inherits empty analyses from template with no recommendations
  Given a requirement template "Minimal" exists with recommended_analyses empty
  And the Requirement Author is creating a new requirement
  And the author selects the template "Minimal"
```

```
  When the author saves the requirement
  Then the requirement's recommended_analyses field is empty
  And the requirement is persisted to the database

@error-path @validation @invalid-input @automation-ready @test:required
Scenario: Author cannot save requirement with unregistered step-code
  Given the Requirement Author is creating a new requirement
  And the author selects the template "Standard Functional"
  And the recommended_analyses field is pre-populated with "R-D,R-F,R-C"
  When the author changes the recommended_analyses field to "R-D,R-INVALID,R-C"
  And the author attempts to save the requirement
  Then the save operation is rejected
  And the system displays an error message "Unknown step-code: R-INVALID"
  And the requirement is not persisted to the database

@error-path @validation @invalid-input @automation-ready @test:required
Scenario: Author cannot save requirement with multiple unregistered step-codes
  Given the Requirement Author is creating a new requirement
  And the author selects the template "Standard Functional"
  When the author changes the recommended_analyses field to "R-UNKNOWN,R-FAKE"
  And the author attempts to save the requirement
  Then the save operation is rejected
  And the system displays an error message identifying the invalid codes "R-UNKNOWN, R-FAKE"
  And the requirement is not persisted to the database

@data-integrity @automation-ready @test:required
Scenario: Recommended analyses are versioned in requirements history
  Given a requirement exists with recommended_analyses "R-D,R-F,R-C"
  When the Requirement Author edits the requirement
  And the author changes the recommended_analyses field to "R-D,R-V"
  And the author saves the requirement
  Then the current requirement record contains recommended_analyses "R-D,R-V"
  And the requirements_history table contains a record with the previous value "R-D,R-F,R-C"
  And the requirements_history table contains a record with the new value "R-D,R-V"

@edge-case @automation-ready @test:required
Scenario: Template selection occurs after requirement form is partially filled
  Given the Requirement Author is creating a new requirement
  And the author has entered a requirement title "User Authentication"
  And the author has not yet selected a template
  When the author selects the template "Security Focused"
  Then the recommended_analyses field is populated with "R-D,R-F,R-V"
  And the requirement title remains "User Authentication"
  And the requirement is not yet saved

@alternate-path @automation-ready @test:required
Scenario: Author clears inherited recommended analyses before saving
  Given the Requirement Author is creating a new requirement
  And the author selects the template "Standard Functional"
  And the recommended_analyses field is pre-populated with "R-D,R-F,R-C"
  When the author clears the recommended_analyses field
  And the author saves the requirement
  Then the requirement's recommended_analyses field is empty
  And the requirement is persisted to the database

Rule: Integration -- Analysis kickoff UI consumes seeded recommended_analyses (FR-18.3.3)

  @integration @kickoff-ui @automation-ready @test:required
  Scenario: Kickoff UI pre-ticks seeded recommended analyses on a newly created requirement
```

```
    Given a requirement "REQ-400" has just been created from a template seeding
  recommended_analyses with "R-D,R-F"
    When the analyst opens the analysis kickoff UI for requirement "REQ-400"
    Then the kickoff UI checkbox list pre-ticks "R-D, R-F"
    And the analyst can adjust the selection before running

  @integration @kickoff-ui @edge-case @automation-ready @test:required
  Scenario: Kickoff UI handles requirement created without a template (no seeded analyses)
    Given a requirement "REQ-401" was created without a template
    And the requirement has empty recommended_analyses
    When the analyst opens the analysis kickoff UI for requirement "REQ-401"
    Then the kickoff UI checkbox list shows all registered analyses unticked
    And the analyst manually selects analyses to run

  @integration @kickoff-ui @edge-case @automation-ready @test:required
  Scenario: Kickoff UI silently skips invalid step-codes carried in seeded recommended_analyses
    Given a requirement "REQ-402" has recommended_analyses "R-D,R-GHOST,R-F"
    And "R-GHOST" is not registered in the analyser step-code registry
    When the analyst opens the analysis kickoff UI for requirement "REQ-402"
    Then the kickoff UI pre-ticks "R-D, R-F"
    And displays a non-blocking notice naming the unrecognised step-code "R-GHOST"

Rule: Seeding timing -- on save, not on form-load (decision 2026-04-27)

  @timing @save-time @automation-ready @test:required
  Scenario: Recommended analyses are seeded at save, using the template active at save time
    Given the user is creating a new requirement in the requirement add form
    And the user picks template "Functional -- Standard" (recommended_analyses "R-D,R-F,R-C")
    And then changes the template selection to "Functional -- Lite" (recommended_analyses
  "R-D,R-F")
    And then saves the requirement
    When the save is committed
    Then the requirement's recommended_analyses is seeded as "R-D,R-F" (from the template
  selected at save time)
    And no intermediate seeding occurred during template selection on the form

  @timing @no-flicker @automation-ready @test:required
  Scenario: Form-time template changes do not trigger any seeding side effects
    Given the user is creating a new requirement in the requirement add form
    And the form has not yet been submitted
    When the user changes the template selection multiple times before saving
    Then no recommended_analyses is written to any database row at this point
    And no preview-flicker behaviour exists (the form does not display "what-if" RA values
  during selection)
```

Validate Recommended Analyses Against Registered Step-Codes

```
@feature:validate-recommended-analyses-against-registered-s @req:FR-18.3.2 @priority:medium @
    test-characteristic:validation @test-characteristic:data-integrity @risk:data-loss
Feature: Validate Recommended Analyses Against Registered Step-Codes

  As a Requirement Author
  I want the system to validate my recommended_analyses selections against the registered
    analyser step-codes
  So that I cannot save invalid or unknown analysis codes and prevent configuration errors that
    would cause analysis runs to fail

  # Duplicate step-code handling: duplicates within a single recommended_analyses
  # value (e.g. "R-D,R-D,R-F") are silently de-duplicated by the validator before
```

```
# storage. The stored canonical form contains each step-code at most once.
# Whitespace handling: leading/trailing whitespace around individual step-codes
# is trimmed silently before validation. Empty or whitespace-only codes between
# commas (e.g. "R-D, , R-F") are rejected as malformed input.
Background:
  Given the analyser step-code registry contains the following registered codes:
    | step_code | description           |
    | R-D       | Design Analysis       |
    | R-F       | Feasibility Analysis  |
    | R-C       | Compliance Analysis   |
    | R-T       | Traceability Analysis |
    | S-V       | Story Validation      |

Rule: Valid step-codes must exist in the registered analyser step-code registry

  @happy-path @automation-ready @test:required
  Scenario: Save requirement with valid single recommended analysis
    Given a requirement author is editing requirement "REQ-1001"
    And the requirement has no recommended_analyses set
    When the author sets recommended_analyses to "R-D"
    And the author saves the requirement
    Then the requirement is saved successfully
    And the recommended_analyses field contains "R-D"

  @happy-path @automation-ready @test:required
  Scenario: Save requirement with multiple valid recommended analyses
    Given a requirement author is editing requirement "REQ-1002"
    And the requirement has no recommended_analyses set
    When the author sets recommended_analyses to "R-D,R-F,R-C"
    And the author saves the requirement
    Then the requirement is saved successfully
    And the recommended_analyses field contains "R-D,R-F,R-C"

  @error-path @validation @automation-ready @test:required
  Scenario: Reject requirement with single unknown step-code
    Given a requirement author is editing requirement "REQ-1003"
    When the author sets recommended_analyses to "R-X"
    And the author attempts to save the requirement
    Then the save operation fails
    And the system displays an error message "Unknown analysis code: R-X"
    And the requirement is not saved

  @error-path @validation @automation-ready @test:required
  Scenario: Reject requirement with mix of valid and invalid step-codes
    Given a requirement author is editing requirement "REQ-1004"
    When the author sets recommended_analyses to "R-D,R-INVALID,R-F"
    And the author attempts to save the requirement
    Then the save operation fails
    And the system displays an error message "Unknown analysis code: R-INVALID"
    And the requirement is not saved

  @error-path @validation @automation-ready @test:required
  Scenario: Reject requirement with multiple unknown step-codes
    Given a requirement author is editing requirement "REQ-1005"
    When the author sets recommended_analyses to "R-X,R-Y,R-Z"
    And the author attempts to save the requirement
    Then the save operation fails
    And the system displays an error message identifying all invalid codes "Unknown analysis
   codes: R-X, R-Y, R-Z"
```

```
    And the requirement is not saved

Rule: Empty recommended_analyses values are permitted

  @edge-case @empty-input @automation-ready @test:required
  Scenario: Save requirement with empty recommended_analyses
    Given a requirement author is editing requirement "REQ-1006"
    When the author sets recommended_analyses to an empty value
    And the author saves the requirement
    Then the requirement is saved successfully
    And the recommended_analyses field is empty

  @edge-case @empty-input @automation-ready @test:required
  Scenario: Save requirement with null recommended_analyses
    Given a requirement author is editing requirement "REQ-1007"
    When the author sets recommended_analyses to null
    And the author saves the requirement
    Then the requirement is saved successfully
    And the recommended_analyses field is null

  @alternate-path @automation-ready @test:required
  Scenario: Clear previously set recommended_analyses
    Given a requirement author is editing requirement "REQ-1008"
    And the requirement has recommended_analyses set to "R-D,R-F"
    When the author clears the recommended_analyses field
    And the author saves the requirement
    Then the requirement is saved successfully
    And the recommended_analyses field is empty

Rule: Validation applies only to requirements, not other artifact types

  @boundary @automation-ready @test:required
  Scenario: Validation enforced for requirement artifacts
    Given a requirement author is editing a requirement artifact "REQ-2001"
    When the author sets recommended_analyses to "R-INVALID"
    And the author attempts to save the artifact
    Then the save operation fails
    And the system displays an error message "Unknown analysis code: R-INVALID"

  @edge-case @automation-ready @test:required
  Scenario: Validation not enforced for story artifacts
    Given a requirement author is editing a story artifact "STORY-3001"
    When the author sets a field to "R-INVALID"
    And the author saves the artifact
    Then the artifact is saved successfully
    And no validation error is raised for analysis codes

Rule: Step-code validation is case-sensitive

  @error-path @validation @boundary @automation-ready @test:required
  Scenario: Reject step-code with incorrect case
    Given a requirement author is editing requirement "REQ-1009"
    When the author sets recommended_analyses to "r-d"
    And the author attempts to save the requirement
    Then the save operation fails
    And the system displays an error message "Unknown analysis code: r-d"

  @error-path @validation @boundary @automation-ready @test:required
  Scenario: Reject step-code with mixed case
```

```
    Given a requirement author is editing requirement "REQ-1010"
    When the author sets recommended_analyses to "R-d,R-F"
    And the author attempts to save the requirement
    Then the save operation fails
    And the system displays an error message "Unknown analysis code: R-d"

Rule: Whitespace handling in comma-separated step-codes

  @edge-case @automation-ready @test:required
  Scenario: Accept step-codes with spaces after commas
    Given a requirement author is editing requirement "REQ-1011"
    When the author sets recommended_analyses to "R-D, R-F, R-C"
    And the author saves the requirement
    Then the requirement is saved successfully
    And the recommended_analyses field contains "R-D, R-F, R-C"

  @edge-case @automation-ready @test:required
  Scenario: Accept step-codes with no spaces after commas
    Given a requirement author is editing requirement "REQ-1012"
    When the author sets recommended_analyses to "R-D,R-F,R-C"
    And the author saves the requirement
    Then the requirement is saved successfully
    And the recommended_analyses field contains "R-D,R-F,R-C"

Rule: Duplicate step-codes are handled gracefully

  @edge-case @automation-ready @test:required
  Scenario: Accept duplicate valid step-codes
    Given a requirement author is editing requirement "REQ-1013"
    When the author sets recommended_analyses to "R-D,R-F,R-D"
    And the author saves the requirement
    Then the requirement is saved successfully
    And the recommended_analyses field contains "R-D,R-F,R-D"

Rule: Retro-placeholder requirements may have empty recommended_analyses

  @edge-case @automation-ready @test:required
  Scenario: Retro-placeholder requirement with empty recommended_analyses
    Given a requirement author is editing requirement "REQ-RETRO-001"
    And the requirement has origin set to "retro-placeholder"
    When the author leaves recommended_analyses empty
    And the author saves the requirement
    Then the requirement is saved successfully
    And the recommended_analyses field is empty
    And the requirement is excluded from default analysis runs

Rule: Outcome -- Behaviour when the analyser step-code registry is empty or unavailable

  @edge-case @degraded @automation-ready @test:required
  Scenario: Validation rejects all step-codes when registry is empty
    Given the analyser step-code registry contains no entries
    When the user attempts to set recommended_analyses to "R-D,R-F" on a template or
  requirement
    Then the save operation is rejected
    And the system displays the error message "No registered analyser step-codes available --
  registry is empty"
    And the existing recommended_analyses value (if any) is unchanged

  @edge-case @degraded @automation-ready @test:required
```

```
  Scenario: Validation falls back to a fail-closed posture when the registry is unavailable
    Given the analyser step-code registry is unreachable (transient infrastructure issue)
    When the user attempts to save recommended_analyses
    Then the save operation is rejected
    And the system displays the error message "Cannot validate step-codes -- analyser registry
  unavailable; please retry"
    And no partial write occurs
```

Stories for FR-18.3.3 — Analysis Kickoff Pre-Selection

Pre-populate Analysis Selector from Requirement Metadata

```
@feature:pre-populate-analysis-selector-from-requirement-me @req:FR-18.3.3 @priority:medium @
    test:required
Feature: Pre-populate Analysis Selector from Requirement Metadata

  As an Analyst
  I want the analysis selector checkboxes to be pre-ticked based on the requirement's
    recommended_analyses metadata when I initiate an analysis run
  So that I can quickly start analysis runs with sensible defaults while retaining the
    flexibility to adjust the selection

  Background:
    Given the requirement "FR-18.3.3" exists in the system

  @happy-path @automation-ready @test:required
  Scenario: Analysis selector pre-populates from requirement metadata
    Given the requirement has recommended_analyses metadata "R-F, R-C"
    When the Analyst initiates an analysis run from the requirement
    Then the analysis selector displays checkboxes for all available analyses
    And the "R-F" checkbox is pre-ticked
    And the "R-C" checkbox is pre-ticked
    And all other analysis checkboxes are not pre-ticked

  @happy-path @automation-ready @test:required
  Scenario: Analyst modifies pre-selected analyses before running
    Given the requirement has recommended_analyses metadata "R-F, R-C"
    And the Analyst has initiated an analysis run from the requirement
    And the "R-F" and "R-C" checkboxes are pre-ticked
    When the Analyst unticks the "R-F" checkbox
    And the Analyst ticks the "R-D" checkbox
    And the Analyst confirms the analysis run
    Then the system executes analyses "R-D, R-C"
    And the analyses row logs the executed selection as "R-D, R-C"
    And the analyses row does not log "R-F"

  @edge-case @empty-input @automation-ready @test:required
  Scenario: No analyses pre-selected when metadata is empty
    Given the requirement has empty recommended_analyses metadata
    When the Analyst initiates an analysis run from the requirement
    Then the analysis selector displays checkboxes for all available analyses
    And no analysis checkboxes are pre-ticked

  @edge-case @empty-input @automation-ready @test:required
  Scenario: No analyses pre-selected when metadata is missing
    Given the requirement has no recommended_analyses metadata field
    When the Analyst initiates an analysis run from the requirement
    Then the analysis selector displays checkboxes for all available analyses
    And no analysis checkboxes are pre-ticked
```

```
@happy-path @data-integrity @automation-ready @test:required
Scenario: Final selection overrides metadata in execution log
  Given the requirement has recommended_analyses metadata "R-F"
  And the Analyst has initiated an analysis run from the requirement
  And the "R-F" checkbox is pre-ticked
  When the Analyst ticks the "R-C" checkbox
  And the Analyst confirms the analysis run
  Then the system executes analyses "R-F, R-C"
  And the analyses row logs the executed selection as "R-F, R-C"
  And the requirement's recommended_analyses metadata remains unchanged as "R-F"

@alternate-path @automation-ready @test:required
Scenario: Analyst deselects all pre-ticked analyses
  Given the requirement has recommended_analyses metadata "R-F, R-C, R-D"
  And the Analyst has initiated an analysis run from the requirement
  And the "R-F", "R-C", and "R-D" checkboxes are pre-ticked
  When the Analyst unticks all pre-selected checkboxes
  And the Analyst confirms the analysis run
  Then the system executes no analyses
  And the analyses row logs an empty executed selection

@happy-path @automation-ready @test:required
Scenario: Multiple requirements with different metadata maintain independent selections
  Given the requirement "FR-18.3.1" has recommended_analyses metadata "R-D"
  And the requirement "FR-18.3.2" has recommended_analyses metadata "R-F, R-C"
  When the Analyst initiates an analysis run from requirement "FR-18.3.1"
  Then only the "R-D" checkbox is pre-ticked
  When the Analyst initiates an analysis run from requirement "FR-18.3.2"
  Then only the "R-F" and "R-C" checkboxes are pre-ticked

@edge-case @validation @automation-ready @test:required
Scenario: Invalid analysis codes in metadata are ignored during pre-selection
  Given the requirement has recommended_analyses metadata "R-F, INVALID-CODE, R-C"
  When the Analyst initiates an analysis run from the requirement
  Then the "R-F" checkbox is pre-ticked
  And the "R-C" checkbox is pre-ticked
  And no checkbox for "INVALID-CODE" is displayed

@happy-path @access-control @automation-ready @test:required
Scenario: Pre-selection respects operator's ability to modify selection
  Given the requirement has recommended_analyses metadata "R-F"
  And the Analyst has initiated an analysis run from the requirement
  When the Analyst attempts to untick the "R-F" checkbox
  Then the "R-F" checkbox becomes unticked
  And the system does not prevent the modification

@data-integrity @automation-ready @test:required
Scenario: Metadata remains unchanged after analysis execution
  Given the requirement has recommended_analyses metadata "R-F, R-C"
  And the Analyst has initiated an analysis run from the requirement
  When the Analyst unticks all checkboxes
  And the Analyst ticks the "R-D" checkbox
  And the Analyst confirms the analysis run
  And the system completes the analysis execution
  Then the requirement's recommended_analyses metadata remains "R-F, R-C"
  And the analyses row logs the executed selection as "R-D"
```

Stories for FR-18.3.5 — Initial Gold-Template RA Population

Migration Report Generation

```
@feature:migration-report-generation @req:FR-18.3.5 @priority:medium @test:required
Feature: Migration Report Generation

  As a System Administrator
  I want a summary report of the migration execution
  So that I can verify the migration completed successfully and retain an audit record of
    changes made

  Background:
    Given the migration script "FR-18.3.5_initial_gold_template_ra_population" has been executed
    And the migration targets orgid 1 with appid 30

  @happy-path @automation-ready @data-integrity
  Scenario: Complete migration report includes all row counts
    Given 27 template records exist in req_templates for orgid 1
    And 150 requirement records exist in appid 30
    And 120 of those requirements have non-empty template associations
    And 80 of those requirements have empty recommended_analyses
    And 10 requirements have origin "retro-placeholder"
    When the migration completes successfully
    Then the migration report is generated
    And the report includes 27 templates updated
    And the report includes 70 requirements seeded
    And the report includes 10 requirements skipped as retro-placeholders
    And the report includes 30 requirements skipped with existing RAs
    And the report is stored in "documentation/migrations/" directory

  @happy-path @automation-ready @audit-trail
  Scenario: Migration report provides org and app breakdown
    Given 27 template records exist for orgid 1
    And 150 requirement records exist across 3 apps in orgid 1
    And appid 30 has 100 requirements eligible for seeding
    And appid 31 has 30 requirements eligible for seeding
    And appid 32 has 20 requirements eligible for seeding
    When the migration completes successfully
    Then the migration report includes a breakdown by org
    And the report shows orgid 1 with 27 templates updated
    And the report includes a breakdown by app
    And the report shows appid 30 with its seeded requirement count
    And the report shows appid 31 with its seeded requirement count
    And the report shows appid 32 with its seeded requirement count

  @happy-path @automation-ready @audit-trail
  Scenario: Migration report retained alongside migration script
    Given the migration script is stored at
    "documentation/migrations/FR-18.3.5_initial_gold_template_ra_population.sql"
    When the migration completes successfully
    And the migration report is generated
    Then the report file is stored in "documentation/migrations/" directory
    And the report filename includes "FR-18.3.5_initial_gold_template_ra_population"
    And the report filename includes a timestamp
    And the report is retained permanently per rolling-migration policy

  @edge-case @automation-ready @empty-input
  Scenario: Migration report handles zero requirements seeded
```

```
  Given 27 template records exist in req_templates for orgid 1
  And all requirements in appid 30 already have non-empty recommended_analyses
  When the migration completes successfully
  Then the migration report is generated
  And the report includes 27 templates updated
  And the report includes 0 requirements seeded
  And the report includes 0 requirements skipped as retro-placeholders
  And the report confirms migration completed successfully

@edge-case @automation-ready @data-integrity
Scenario: Migration report distinguishes skipped categories
  Given 27 template records exist in req_templates for orgid 1
  And 100 requirement records exist in appid 30
  And 20 requirements have origin "retro-placeholder"
  And 30 requirements have no template association
  And 25 requirements already have non-empty recommended_analyses
  And 25 requirements are eligible for seeding
  When the migration completes successfully
  Then the migration report is generated
  And the report includes 25 requirements seeded
  And the report includes 20 requirements skipped as retro-placeholders
  And the report includes 30 requirements skipped with no template
  And the report includes 25 requirements skipped with existing RAs
  And the report total accounts for all 100 requirements

@happy-path @automation-ready @data-integrity
Scenario: Migration report confirms template cascade completion
  Given 27 Gold template files exist in "static/templates/reqs_templates/"
  And each Gold file has a reviewed recommended_analyses front-matter key
  And 27 corresponding template records exist in req_templates for orgid 1
  When the migration completes successfully
  Then the migration report confirms 27 templates updated
  And the report confirms each template recommended_analyses matches its Gold file
  And the report includes no template update failures

@error-path @automation-ready @data-integrity
Scenario: Migration report captures partial completion on error
  Given 27 template records exist in req_templates for orgid 1
  And 150 requirement records exist in appid 30
  And the migration updates 27 templates successfully
  And the migration seeds 50 requirements successfully
  When the migration encounters an error during requirement seeding
  Then the migration report is generated
  And the report includes 27 templates updated
  And the report includes 50 requirements seeded before error
  And the report includes an error message describing the failure
  And the report indicates migration did not complete fully

@happy-path @automation-ready @audit-trail
Scenario: Migration report includes execution timestamp
  Given the migration script begins execution at "2025-01-15T14:30:00Z"
  And the migration completes at "2025-01-15T14:32:15Z"
  When the migration report is generated
  Then the report includes execution start timestamp "2025-01-15T14:30:00Z"
  And the report includes execution end timestamp "2025-01-15T14:32:15Z"
  And the report includes total execution duration "2 minutes 15 seconds"

@happy-path @automation-ready @verification
Scenario: Migration report provides sufficient detail for verification
```

```
  Given the migration completes successfully
  When the migration report is generated
  Then the report includes total templates updated count
  And the report includes total requirements seeded count
  And the report includes total requirements skipped count with reason breakdown
  And the report includes orgid and appid scope
  And the report includes execution timestamps
  And the report provides sufficient information to verify migration success
  And the report supports audit requirements

@alternate-path @automation-ready @audit-trail
Scenario: Migration report delivered as console output
  Given the migration script is executed from command line
  When the migration completes successfully
  Then the migration report is written to console output
  And the console output includes all required report elements
  And the console output is also captured to a log file
  And the log file is stored in "documentation/migrations/" directory

@alternate-path @automation-ready @audit-trail
Scenario: Migration report delivered as database record
  Given the system has a migration_log table
  When the migration completes successfully
  Then a migration_log record is created
  And the record includes migration script identifier
  "FR-18.3.5_initial_gold_template_ra_population"
  And the record includes templates_updated count
  And the record includes requirements_seeded count
  And the record includes requirements_skipped count with reason breakdown
  And the record includes execution timestamps
  And the record includes migration status "completed"

@edge-case @automation-ready @data-integrity
Scenario: Migration report handles requirements without templates
  Given 27 template records exist in req_templates for orgid 1
  And 150 requirement records exist in appid 30
  And 50 requirements have no template association
  And 100 requirements have template associations
  When the migration completes successfully
  Then the migration report is generated
  And the report includes 50 requirements skipped with no template
  And the report confirms requirements without templates were left untouched
  And the report notes these are tracked for separate data-hygiene cleanup

Rule: Report format and delivery (decisions 2026-04-27)

  @report @format @automation-ready @test:required
  Scenario: Report is generated as a Markdown file written next to the migration script
    Given the migration script has run to completion (or partial completion)
    When the run finishes
    Then a Markdown file is written at
  "documentation/migrations/0.X/populate_template_ras_report_<UTC_timestamp>.md"
    And the same content is also printed to the Claude Code console (stdout) when the script
   is invoked from a Claude Code session

  @report @structure @automation-ready @test:required
  Scenario: Report includes a header summary and a per-row status table
    Given the script processed templates and/or requirements
    When the report is generated
```

```
    Then the header summarises counts per status, broken down by phase: "templates" and
  "requirements"
    And the body contains one row per processed template and per processed requirement
    And each row carries: identifier, status, reason (if not "applied")

  @report @status-enum @automation-ready @test:required
  Scenario: Each row carries exactly one status from the fixed enum
    Given a row in the report describes one processed entity
    When the status is recorded
    Then it is one of: "applied" (value written), "skipped" (deliberately not processed),
  "failed" (error during processing), or "unchanged" (target already matched the desired value)
    And no row carries multiple statuses

  @report @skip-precedence @automation-ready @test:required
  Scenario: Skip categorisation uses fixed precedence -- first match wins
    Given a requirement is eligible for several skip reasons simultaneously
    When the script categorises the skip
    Then it applies this fixed precedence (top-down, first match wins):
      1. retro-placeholder (origin='retro-placeholder')
      2. no-template (template_id is NULL)
      3. already-populated (recommended_analyses is non-empty)
      4. template-not-yet-populated (template's RAs is empty)
    And exactly one reason is recorded on the row

  @report @console-output @automation-ready @test:required
  Scenario: Console output mirrors the Markdown file
    Given the script is invoked from a Claude Code session
    When the report is generated
    Then the Claude Code console receives the same Markdown content as the file
    And no information is in one but not the other
```

Requirement Recommended Analyses Seeding

```
@feature:requirement-recommended-analyses-seeding @req:FR-18.3.5 @priority:medium @
    risk:data-loss @data-integrity
Feature: Requirement Recommended Analyses Seeding

  As a System Administrator
  I want to seed recommended_analyses values on existing REQQA app requirements from their
    templates
  So that the analysis kickoff UI provides sensible defaults for requirements created before
    FR-18.3.2 was delivered

  Background:
    Given the REQQA application exists with appid 30 and orgid 1
    And the Template Recommended Analyses Population feature has been completed
    And the req_templates table contains template records with populated recommended_analyses
    values

  @happy-path @automation-ready @test:required
  Scenario: Seed requirement with empty RAs from template with non-empty RAs
    Given a requirement exists in appid 30 with the following attributes:
      | attribute              | value                |
      | requirement_id         | REQ-001              |
      | template_id            | TMPL-FUNC-01         |
      | recommended_analyses   |                      |
      | origin                 | user-created         |
    And the template "TMPL-FUNC-01" has recommended_analyses "R-D, R-F, R-C"
    When the requirement recommended analyses seeding operation executes
```

```
  Then requirement "REQ-001" has recommended_analyses "R-D, R-F, R-C"
  And the requirement's other attributes remain unchanged

@happy-path @automation-ready @test:required
Scenario: Seed multiple requirements from the same template
  Given the following requirements exist in appid 30:
    | requirement_id | template_id  | recommended_analyses | origin       |
    | REQ-100        | TMPL-NFR-02  |                      | user-created |
    | REQ-101        | TMPL-NFR-02  |                      | user-created |
    | REQ-102        | TMPL-NFR-02  |                      | user-created |
  And the template "TMPL-NFR-02" has recommended_analyses "R-P, R-S"
  When the requirement recommended analyses seeding operation executes
  Then all 3 requirements have recommended_analyses "R-P, R-S"

@edge-case @automation-ready @test:required
Scenario: Skip requirement with retro-placeholder origin
  Given a requirement exists in appid 30 with the following attributes:
    | attribute              | value                |
    | requirement_id         | REQ-RETRO-01         |
    | template_id            | TMPL-FUNC-01         |
    | recommended_analyses   |                      |
    | origin                 | retro-placeholder    |
  And the template "TMPL-FUNC-01" has recommended_analyses "R-D, R-F, R-C"
  When the requirement recommended analyses seeding operation executes
  Then requirement "REQ-RETRO-01" has recommended_analyses ""
  And the requirement remains unchanged

@edge-case @automation-ready @test:required
Scenario: Skip requirement with existing non-empty RAs
  Given a requirement exists in appid 30 with the following attributes:
    | attribute              | value                |
    | requirement_id         | REQ-200              |
    | template_id            | TMPL-FUNC-01         |
    | recommended_analyses   | R-D, R-X             |
    | origin                 | user-created         |
  And the template "TMPL-FUNC-01" has recommended_analyses "R-D, R-F, R-C"
  When the requirement recommended analyses seeding operation executes
  Then requirement "REQ-200" has recommended_analyses "R-D, R-X"
  And the requirement's recommended_analyses value is not overwritten

@edge-case @automation-ready @test:required
Scenario: Skip requirement when template has empty RAs
  Given a requirement exists in appid 30 with the following attributes:
    | attribute              | value                |
    | requirement_id         | REQ-300              |
    | template_id            | TMPL-EMPTY-01        |
    | recommended_analyses   |                      |
    | origin                 | user-created         |
  And the template "TMPL-EMPTY-01" has recommended_analyses ""
  When the requirement recommended analyses seeding operation executes
  Then requirement "REQ-300" has recommended_analyses ""
  And the requirement remains unchanged

@edge-case @automation-ready @test:required
Scenario: Skip requirement without an associated template
  Given a requirement exists in appid 30 with the following attributes:
    | attribute              | value                |
    | requirement_id         | REQ-ORPHAN-01        |
    | template_id            |                      |
```

```
    | recommended_analyses   |                      |
    | origin                 | user-created         |
  When the requirement recommended analyses seeding operation executes
  Then requirement "REQ-ORPHAN-01" has recommended_analyses ""
  And the requirement remains unchanged

@boundary @automation-ready @test:required
Scenario: Seed requirement in REQQA app only, not other apps
  Given a requirement exists in appid 30 with the following attributes:
    | attribute              | value                |
    | requirement_id         | REQ-REQQA-01         |
    | template_id            | TMPL-FUNC-01         |
    | recommended_analyses   |                      |
    | origin                 | user-created         |
  And a requirement exists in appid 99 with the following attributes:
    | attribute              | value                |
    | requirement_id         | REQ-OTHER-01         |
    | template_id            | TMPL-FUNC-01         |
    | recommended_analyses   |                      |
    | origin                 | user-created         |
  And the template "TMPL-FUNC-01" has recommended_analyses "R-D, R-F, R-C"
  When the requirement recommended analyses seeding operation executes
  Then requirement "REQ-REQQA-01" in appid 30 has recommended_analyses "R-D, R-F, R-C"
  And requirement "REQ-OTHER-01" in appid 99 has recommended_analyses ""

@happy-path @large-dataset @automation-ready @test:required
Scenario: Seed all eligible requirements across multiple templates
  Given 50 requirements exist in appid 30 with empty recommended_analyses
  And these requirements are distributed across 10 different templates
  And all 10 templates have non-empty recommended_analyses values
  And none of the requirements have origin "retro-placeholder"
  When the requirement recommended analyses seeding operation executes
  Then all 50 requirements have recommended_analyses values matching their respective templates
  And no requirements outside appid 30 are modified

@edge-case @idempotency @automation-ready @test:required
Scenario: Re-running seeding operation does not modify already-seeded requirements
  Given a requirement exists in appid 30 with the following attributes:
    | attribute              | value                |
    | requirement_id         | REQ-400              |
    | template_id            | TMPL-FUNC-01         |
    | recommended_analyses   |                      |
    | origin                 | user-created         |
  And the template "TMPL-FUNC-01" has recommended_analyses "R-D, R-F, R-C"
  And the requirement recommended analyses seeding operation has already executed once
  And requirement "REQ-400" now has recommended_analyses "R-D, R-F, R-C"
  When the requirement recommended analyses seeding operation executes again
  Then requirement "REQ-400" still has recommended_analyses "R-D, R-F, R-C"
  And the requirement's recommended_analyses value is not duplicated or modified

@error-path @validation @automation-ready @test:required
Scenario: Seeding fails gracefully when template reference is invalid
  Given a requirement exists in appid 30 with the following attributes:
    | attribute              | value                |
    | requirement_id         | REQ-500              |
    | template_id            | TMPL-INVALID-99      |
    | recommended_analyses   |                      |
    | origin                 | user-created         |
  And the template "TMPL-INVALID-99" does not exist in req_templates
```

```
    When the requirement recommended analyses seeding operation executes
    Then requirement "REQ-500" has recommended_analyses ""
    And the operation logs a warning for the invalid template reference
    And the operation continues processing other requirements

  @happy-path @automation-ready @test:required
  Scenario: Seed requirement with complex multi-step RAs value
    Given a requirement exists in appid 30 with the following attributes:
      | attribute              | value                |
      | requirement_id         | REQ-600              |
      | template_id            | TMPL-COMPLEX-01      |
      | recommended_analyses   |                      |
      | origin                 | user-created         |
    And the template "TMPL-COMPLEX-01" has recommended_analyses "R-D, R-F, R-C, R-P, R-S, R-T"
    When the requirement recommended analyses seeding operation executes
    Then requirement "REQ-600" has recommended_analyses "R-D, R-F, R-C, R-P, R-S, R-T"
    And the full comma-separated list is preserved exactly as in the template

  Rule: Migration mechanism -- execution, atomicity, recovery (decisions 2026-04-27)

    @migration @cli @automation-ready @test:required
    Scenario: Requirement seeding is invoked via the same CLI script as template population
      Given the migration script at "documentation/migrations/0.X/populate_template_ras.py"
    exists (see story 184)
      And the script supports "--phase=requirements" or "--phase=all"
      When an administrator runs the script with "--phase=requirements"
      Then only the requirement-seeding phase runs against requirements whose templates already
    have populated recommended_analyses
      And templates that have not yet been populated cause their dependent requirements to be
    skipped with reason "template-not-yet-populated"

    @migration @atomicity @automation-ready @test:required
    Scenario: Requirement seeding uses per-record commit
      Given a corpus of N requirements eligible for seeding
      When the script processes them sequentially
      Then each requirement's update is committed before the next requirement is processed
      And a failure on one requirement does not roll back prior successful updates
      And the report records each row's outcome individually

    @migration @rollback @automation-ready @test:required
    Scenario: Recovery is by re-run, not rollback
      Given a partial seeding has occurred and the script failed mid-run
      When the underlying issue is fixed and the script is re-run with "--phase=requirements"
      Then requirements that were successfully seeded earlier are skipped with reason
    "already-populated"
      And requirements not yet seeded are processed
      And the run reaches a consistent state without any rollback step
```

Template Recommended Analyses Population

```
@feature:template-recommended-analyses-population @req:FR-18.3.5 @priority:medium @
    risk:data-loss @data-integrity
Feature: Template Recommended Analyses Population

  As a System Administrator
  I want to populate recommended_analyses values on all REQQA org templates
  So that the kickoff UI can pre-select appropriate analysis steps for requirements created from
    those templates
```

```
Background:
  Given the REQQA organisation exists with orgid 1
  And the req_templates table contains 27 template records for orgid 1
  And the recommended_analyses column exists on req_templates

@happy-path @test:required @automation-ready
Scenario: All functional requirement templates receive appropriate analysis step-codes
  Given the template "Functional Requirement - Standard" exists for orgid 1
  And the template has no recommended_analyses value
  When the template recommended analyses population is executed
  Then the template "Functional Requirement - Standard" has recommended_analyses set to "R-D,
  R-F, R-C"
  And the recommended_analyses value is non-null

@happy-path @test:required @automation-ready
Scenario: NFR templates receive appropriate analysis step-codes
  Given the template "Non-Functional Requirement - Performance" exists for orgid 1
  And the template has no recommended_analyses value
  When the template recommended analyses population is executed
  Then the template "Non-Functional Requirement - Performance" has recommended_analyses set to
  "R-D, R-F"
  And the recommended_analyses value is non-null

@happy-path @test:required @automation-ready
Scenario: Test requirement templates receive appropriate analysis step-codes
  Given the template "Test Requirement - Integration" exists for orgid 1
  And the template has no recommended_analyses value
  When the template recommended analyses population is executed
  Then the template "Test Requirement - Integration" has recommended_analyses set to "R-D"
  And the recommended_analyses value is non-null

@happy-path @test:required @automation-ready
Scenario: ISO 29148 templates receive appropriate analysis step-codes
  Given the template "ISO 29148 - Full Format" exists for orgid 1
  And the template has no recommended_analyses value
  When the template recommended analyses population is executed
  Then the template "ISO 29148 - Full Format" has recommended_analyses set to "R-D, R-F, R-C"
  And the recommended_analyses value is non-null

@edge-case @test:required @automation-ready @empty-input
Scenario: Templates with no applicable analyses receive explicit empty string
  Given the template "Legacy Template - Deprecated" exists for orgid 1
  And the template has no recommended_analyses value
  And no analysis step-codes are appropriate for this template type
  When the template recommended analyses population is executed
  Then the template "Legacy Template - Deprecated" has recommended_analyses set to ""
  And the recommended_analyses value is non-null
  And a decision record exists documenting why no analyses apply

@happy-path @test:required @automation-ready @data-integrity
Scenario: All 27 templates are reviewed and assigned values
  Given all 27 template records for orgid 1 have null recommended_analyses
  When the template recommended analyses population is executed
  Then all 27 template records have non-null recommended_analyses values
  And each template has either a comma-separated list of step-codes or an explicit empty string
  And no template has a null recommended_analyses value

@error-path @test:required @automation-ready @validation
Scenario: Population fails when template record is missing
```

```
  Given only 26 of the expected 27 template records exist for orgid 1
  When the template recommended analyses population is executed
  Then the population process reports an error
  And the error message indicates "Expected 27 templates for orgid 1, found 26"
  And no template recommended_analyses values are modified

@edge-case @test:required @automation-ready @idempotency
Scenario: Re-running population on already-populated templates is idempotent
  Given all 27 template records for orgid 1 have non-null recommended_analyses values
  And the template "Functional Requirement - Standard" has recommended_analyses "R-D, R-F, R-C"
  When the template recommended analyses population is executed again
  Then the template "Functional Requirement - Standard" still has recommended_analyses "R-D,
  R-F, R-C"
  And no template recommended_analyses values are changed

Rule: Analysis step-codes must match the analyses.stepcode format

  @validation @test:required @automation-ready
  Scenario: Step-codes follow the required format pattern
    Given the template "Functional Requirement - Standard" exists for orgid 1
    When the template recommended analyses population is executed
    Then the template's recommended_analyses value contains only valid step-codes
    And each step-code matches the pattern "R-[A-Z]"
    And step-codes are comma-separated with spaces

  @error-path @test:required @automation-ready @invalid-input
  Scenario: Invalid step-code format is rejected during population
    Given the template "Functional Requirement - Standard" exists for orgid 1
    And the population logic attempts to assign "INVALID, R-D" as recommended_analyses
    When the template recommended analyses population is executed
    Then the population process reports a validation error
    And the error message indicates "Invalid step-code format: INVALID"
    And the template's recommended_analyses remains null

Rule: Templates for different requirement types receive appropriate analysis combinations

  @boundary @test:required @automation-ready
  Scenario Outline: Each template type receives its designated analysis step-codes
    Given the template "<template_name>" of type "<template_type>" exists for orgid 1
    And the template has no recommended_analyses value
    When the template recommended analyses population is executed
    Then the template "<template_name>" has recommended_analyses set to "<expected_analyses>"

    Examples:
      | template_name                          | template_type | expected_analyses |
      | Functional Requirement - Standard      | functional    | R-D, R-F, R-C     |
      | Functional Requirement - Complex       | functional    | R-D, R-F, R-C     |
      | Non-Functional Requirement - Security  | NFR           | R-D, R-F          |
      | Non-Functional Requirement - Usability | NFR           | R-D, R-F          |
      | Test Requirement - Unit                | TR            | R-D               |
      | Test Requirement - System              | TR            | R-D               |
      | ISO 29148 - Light Format               | 29148         | R-D, R-F, R-C     |
      | ISO 29148 - Full Format                | 29148         | R-D, R-F, R-C     |

@edge-case @test:required @automation-ready @data-integrity
Scenario: Population establishes canonical metadata for requirement inheritance
  Given all 27 template records for orgid 1 have null recommended_analyses
  When the template recommended analyses population is executed
  Then each template has a reviewed recommended_analyses value
```

```
  And the values are ready to be inherited by requirements created from these templates
  And the metadata serves as the canonical source for the requirement seeding feature

@alternate-path @test:required @automation-ready
Scenario: Templates with previously set values are not overwritten without explicit decision
  Given the template "Functional Requirement - Standard" has recommended_analyses "R-D, R-F"
  And the template was previously reviewed and assigned this value
  When the template recommended analyses population is executed
  Then the template "Functional Requirement - Standard" retains recommended_analyses "R-D, R-F"
  And a warning is logged indicating the template was already populated

@error-path @test:required @automation-ready @risk:data-loss
Scenario: Population process rolls back on partial failure
  Given 27 template records exist for orgid 1
  And 20 templates have been successfully assigned recommended_analyses values
  When an error occurs processing the 21st template
  Then all 27 templates have their original recommended_analyses values
  And no partial updates are committed
  And an error report identifies which template caused the failure

Rule: Term -- 'Template recommended analyses population' definition

  @glossary @automation-ready @test:required
  Scenario: 'Template recommended analyses population' is defined for this story
    Given the operational term 'Template recommended analyses population' is referenced in
  this story
    Then it is defined as: the one-off migration activity in which the canonical Gold .txt
  files at static/templates/reqs_templates/ have a recommended_analyses front-matter key
  reviewed and assigned, and those values are then propagated into the corresponding rows in
  the req_templates table for every org that holds a copy of that template
    And it is distinct from per-requirement recommended-analyses seeding (FR-18.3.2)
    And it is distinct from the analysis-kickoff pre-selection consumer (FR-18.3.3)

Rule: Migration mechanism -- execution, identification, and atomicity (decisions 2026-04-27)

  @migration @cli @automation-ready @test:required
  Scenario: Template population is invoked as a CLI migration script
    Given the migration script is located at
  "documentation/migrations/0.X/populate_template_ras.py"
    And the script accepts a "--phase" argument with values "templates", "requirements", or
  "all" (default "all")
    When an administrator runs the script with "--phase=templates"
    Then only the template-population phase runs
    And the requirement-seeding phase is skipped

  @migration @template-identification @automation-ready @test:required
  Scenario: Templates are identified by name match between gold .txt files and req_templates
  rows
    Given the gold templates are .txt files in "static/templates/reqs_templates/"
    And each .txt file carries a "name:" key in its YAML-style front-matter
    And each .txt file carries a "recommended_analyses:" key in its front-matter (assigned per
  FR-18.3.5)
    When the script runs the templates phase
    Then for each gold .txt file, every row in "req_templates" whose "name" matches the file's
  "name" key is updated
    And the match is exact (case-sensitive, whitespace-insensitive on the name)

  @migration @overwrite @automation-ready @test:required
  Scenario: Template population always overwrites the existing recommended_analyses value
```

```
    Given a row in "req_templates" exists for the template "Standard Functional Requirement"
    And the row may or may not have an existing "recommended_analyses" value (irrelevant)
    When the script applies the gold value "R-D,R-F,R-C" for that template
    Then the row's "recommended_analyses" column is set to "R-D,R-F,R-C"
    And no merge or skip-on-existing logic applies

  @migration @atomicity @automation-ready @test:required
  Scenario: Template population uses per-template commit (no global transaction)
    Given the gold corpus contains 27 templates
    And one template's update fails mid-run (e.g. database error)
    When the script processes templates sequentially
    Then templates processed before the failure are committed and persisted
    And templates after the failure are not attempted
    And the report records the failure and the unprocessed templates as "skipped -- pending
  re-run"

  @migration @validation-timing @automation-ready @test:required
  Scenario: Validation happens per template, not as a pre-check across all templates
    Given a gold .txt file's recommended_analyses contains a malformed value
    When the script processes that template
    Then only that template fails ("failed" status with reason)
    And subsequent templates continue to be processed
    And no pre-check is performed before the run begins

  @migration @step-code-assignment @automation-ready @test:required
  Scenario: Step-codes for each template are sourced from the gold .txt files (not decided by
  the script)
    Given the gold .txt files have their "recommended_analyses:" front-matter values
  pre-assigned (per FR-18.3.5)
    When the script runs
    Then it reads the assigned values verbatim and writes them to req_templates rows
    And it does not invent, infer, or recompute step-code assignments

  @migration @rollback @automation-ready @test:required
  Scenario: Recovery from failure is by re-run, not rollback
    Given the script has failed mid-run leaving partial state in req_templates
    When the underlying issue is fixed
    And the script is re-run with "--phase=templates"
    Then the script re-applies the gold value to every template (overwrite always)
    And the final state is consistent with the gold corpus regardless of partial prior state
    And no rollback or backup-restore step is required

  @migration @registry-validation @automation-ready @test:required
  Scenario: Step-codes in gold files are trusted as registered (validation deferred)
    Given the gold .txt files are authored to use only step-codes registered in
  modules/statusutils.py STEP_CODES
    When the script runs the templates phase
    Then it does not separately validate gold-file step-codes against the analyser registry in
  v1
    And future hardening to add registry validation is captured as a backlog item, not part of
  this scope

Rule: Term -- 'Template recommended analyses population' definition (consolidated)

  @glossary @automation-ready @test:required
  Scenario: 'Template recommended analyses population' is the operation defined by this story
    Given the operational term 'Template recommended analyses population' is referenced in
  this story
    Then it is defined as: the one-off CLI migration that reads "recommended_analyses:"
```

```
front-matter from gold .txt files in "static/templates/reqs_templates/" and overwrites the
corresponding "req_templates.recommended_analyses" column for every org's copy of each
template
  And it is distinct from per-requirement recommended-analyses seeding (FR-18.3.2 / story
182)
  And it is distinct from the analysis-kickoff pre-selection consumer (FR-18.3.3)
```

Traceability matrix

Functional requirements in this release and their realising user stories. Non-functional requirements are verified through their own verification methods (see each NFR's Verification Method section) rather than through user stories.

Requirement	Title	Stories
FR-18.3.1	Template Recommended Analyses Metadata	Manage Recommended Analyses on Requirement Templates
FR-18.3.2	Requirement Recommended Analyses	Persist Recommended Analyses in Database Schema; Preserve Recommended Analyses on Template Removal; Prompt User on Template Change with Merge Options; Render Recommended Analyses on Requirement View; Seed Recommended Analyses from Template on Requirement Creation; Validate Recommended Analyses Against Registered Step-Codes
FR-18.3.3	Analysis Kickoff Pre-Selection	Pre-populate Analysis Selector from Requirement Metadata
FR-18.3.5	Initial Gold-Template RA Population	Migration Report Generation; Requirement Recommended Analyses Seeding; Template Recommended Analyses Population

Appendix A — Glossary

Terms referenced in the mission, requirements, and stories of this release scope. Definitions are drawn from the application glossary.

acceptance criteria — A set of verifiable conditions that must be satisfied to confirm successful implementation of a feature, requirement, or system component. These criteria establish measurable thresholds for functionality, performance, security, and compliance, providing an objective basis for validating that delivered capabilities meet stakeholder expectations and operational needs.

accepted — A developer confirmation status indicating that the developer has tested the build deliverables, verified they meet the scope requirements, and formally accepted the build as complete and ready for closure. ACCEPTED status is a prerequisite for executing the /close-build skill and represents the developer's sign-off that the build phase is successfully concluded.

access control — A security mechanism that regulates which users, systems, or API clients can access specific resources, operations, or data within the platform. In the context of API integration, it provides granular permission management through custom API keys, ensuring that external systems can only perform authorized actions and access permitted information while maintaining audit trails of all access attempts.

age — A computed display value representing the elapsed time since a backlog item was created, calculated as the difference between the current timestamp and the item's created_at timestamp, and rendered in human-readable relative time format (e.g., '2 days ago', '3 weeks ago', '1 month ago') to help analysts quickly assess item recency.

analyser — A software component or module within REQQA that executes a specific type of analysis (such as R-D Definitions, R-F Functional, R-C Completeness) on a requirement or story, applying predefined rubrics to identify issues, assess quality, and generate structured feedback. Each analyser corresponds to a step-code and produces analysis results stored in the analyses and analysis_issues tables.

analyser step-code — A short alphanumeric identifier (e.g., 'R-D', 'R-F', 'R-C') that uniquely identifies a specific requirement analysis step or procedure within the REQQA analysis framework. Step-codes follow a consistent naming convention and are registered in the system's analyser registry, enabling references to analysis types in templates, requirements, and audit trails.

analysis engine — The core REQQA subsystem responsible for executing requirement analyses by orchestrating AI calls, managing analysis workflows, detecting issues, tracking progress, and persisting results to the analyses and analysis_issues database tables. Located in the modules/ and harness/ directories, it implements the step-code taxonomy (R-D, R-F, R-C, etc.) and coordinates worker processes to perform asynchronous analysis operations.

analyst — A user role in REQQA with permissions to create and manage requirements, stories, scopes, and backlog items within their organization. Analysts are responsible for requirements specification, scope definition, and responding to builder feedback during the Dark Factory lifecycle.

api — Application Programming Interface - a set of defined methods, protocols, and tools that allow different software applications to communicate and exchange data. In this context, REST APIs expose HTTP endpoints that accept requests in a specified format (typically JSON) and return structured responses, enabling integration between the order management system and external services (payment gateway, fulfillment system, carrier tracking).

application — A software system or product being specified and developed, serving as the top-level organizational container for requirements, stories, and scopes. Each application represents a distinct development effort with its own set of specifications, and scopes cannot span multiple applications. Applications provide the context within which requirements are authored and scopes are defined.

appropriate indexing — Database index structures strategically applied to columns frequently used in query WHERE clauses, JOIN conditions, or ORDER BY operations to optimize query performance. Appropriateness is determined by query patterns, data volume, update frequency, and the trade-off between read performance and write overhead.

artifact — A generic term for any primary entity in the REQQA system that can have associated analyses and issues, including requirements, stories, scopes, and applications. Artifacts are identified by a combination of artifact_type (e.g., 'requirement', 'story', 'scope') and artifact_id (the entity's unique identifier), enabling polymorphic querying of analyses and issues across different entity types.

asynchronous — A processing pattern where email sending operations are decoupled from the user request-response cycle, allowing the system to accept the notification request, queue it for later processing, and immediately return control to the calling operation without waiting for email delivery completion. This prevents slow email operations from blocking user-facing transactions and improves system responsiveness.

available — A book status indicating that the item is not currently on loan, not on hold for another member, and is physically present in the library or digitally accessible for immediate borrowing. An available book can be checked out by any eligible member or allocated to the first member in the reservation queue.

backlog item — A structured work item representing a future task, enhancement, defect, or change that has been identified during scope review, design, or build phases but falls outside the current scope. Backlog items are stored per organization and application, linked to source artifacts, prioritized for future implementation, and may be promoted to requirements or stories in subsequent scopes through manual workflow.

bulk-propagation admin action — An administrative operation that applies template changes (specifically recommended_analyses updates) to multiple existing requirements in a single transaction, typically performed by system administrators when a template's analysis recommendations are updated and need to be synchronized across all requirements using that template.

canonical master — The authoritative, reference version of a data entity (such as a template) that serves as the source of truth for replication or inheritance across multiple instances. A canonical master defines the standard structure and content that derivative instances should follow, and changes to the canonical master may propagate to dependent instances according to system rules.

client — The user-facing application or interface (such as a web browser, mobile app, or desktop application) that initiates requests to the server and presents information to the end user. The client runs on the user's device and communicates with the server over a network.

comprehensive — In the context of audit trails, capturing all relevant details necessary to reconstruct the complete state and history of a transaction or change, including who performed the action, when it occurred, what was changed (before and after values), why it was changed (if applicable), and the context in which it occurred.

concurrent users — The number of authenticated users actively interacting with the system simultaneously within a defined time window (typically measured as users with active sessions performing transactions within a 1-minute interval). This differs from total logged-in users, as it counts only those actively generating system load through requests or operations.

create — The action of adding a new investment holding record to the user's portfolio by specifying all required attributes (asset identifier, quantity, acquisition date, cost basis). Creation establishes a new holding that did not previously exist in the system and immediately affects portfolio calculations.

decision — A discrete technical choice made during the design, review, planning, or build phase that addresses a specific aspect of system implementation. Each decision documents: (1) the decision made, (2) alternative options considered, (3) rationale for the chosen approach, (4) requirements affected, and (5) current status (proposed, accepted, rejected). Decisions are logged individually within a scope and require developer approval before being considered final.

deep link —

delete — The action of removing an investment holding record from the user's active portfolio, making it no longer visible in current holdings or included in portfolio calculations. Deletion may be logical (marking as inactive while preserving historical data) or physical (permanent removal), depending on audit and tax reporting requirements.

delivered — An order status indicating that the package has been successfully handed over to the customer or an authorized recipient at the delivery address, or left in a secure location according to delivery instructions. This status is typically confirmed by carrier delivery confirmation (signature, photo, or GPS verification) and represents the terminal successful state of an order.

dependency — A priority classification for backlog items indicating that the item is required as a prerequisite for other planned work or represents technical infrastructure needed to support future features. Dependency items may not deliver direct user value but are necessary to unblock or enable other development efforts. This is the second-highest priority classification in the product backlog taxonomy.

design decision — A discrete technical choice made during the design phase that addresses a specific aspect of the system implementation. Each design decision documents: (1) the decision made, (2) alternative options considered, (3) rationale for the chosen approach, and (4) current status (proposed, approved, rejected, superseded). Design decisions are logged individually within the design record and require developer approval before being considered final.

developer — In the context of scope management, the user role responsible for creating and defining scopes, analyzing requirements, and preparing scopes for handover to builders. The developer (scope creator or organization member with appropriate permissions) has exclusive authority to transition scope status and revise scopes based on builder feedback. This role is distinct from software developers who write code.

error message — A user-facing notification displayed when an operation fails or encounters an invalid state, providing a clear explanation of what went wrong and actionable guidance for resolution. Error messages must be non-technical, specific to the failure context (e.g., 'Basket storage limit reached. Please remove items or proceed to checkout.'), and include an error code for support reference where applicable.

exponential backoff — A retry strategy where the wait time between retry attempts increases exponentially (e.g., 1s, 2s, 4s, 8s) to avoid overwhelming a failing service while giving it time to recover. Often combined with jitter (random variation) to prevent synchronized retry storms from multiple clients.

gold template — A canonical, authoritative template record that serves as the master reference for a particular requirement type across all organizations in the system. Gold templates define the standard structure, recommended analyses, and metadata that should be replicated when organizations create their own template instances. In the current system, the REQQA org's template set (orgid = 1) serves as the de-facto Gold template collection.

gptlog — A logging and accounting subsystem within REQQA that records all calls made to OpenAI's API, capturing request parameters, response data, token consumption (prompt tokens, completion tokens, total tokens), timestamps, costs, and call outcomes. The gptlog provides an audit trail of AI interactions, enables token usage monitoring and cost tracking, and supports debugging of analysis quality issues by preserving the full context of each AI invocation.

holding — A record representing a specific quantity of a financial asset owned by a user in their investment portfolio, including the asset identifier, quantity held, acquisition date, and cost basis per unit. Holdings are the fundamental unit of portfolio composition and are used to calculate total portfolio value, performance metrics, and tax implications.

i18n — Internationalization (abbreviated as i18n, where 18 represents the number of letters between 'i' and 'n') - the process of designing software to support multiple languages, locales, and cultural conventions without requiring code changes. i18n includes support for translated text, date/time formats, number formats, currency, and text direction.

idempotency — A property of an operation where executing it multiple times with the same input produces the same result as executing it once, with no additional side effects. In the context of basket operations, idempotency ensures that duplicate API requests (e.g., due to network retries) do not create duplicate basket items or apply duplicate discounts. Implemented using idempotency keys or request deduplication mechanisms.

immutable — A data attribute or record that cannot be modified after creation. Immutable fields are

write-once, ensuring data integrity and supporting audit requirements by preventing tampering with historical records. Attempts to update immutable fields should be rejected by the system with an error.

inactivity — A period during which no basket-related events occur for a registered user, measured as the absence of basket view, item add, item remove, quantity modification, or checkout initiation events. Used to determine when a basket transitions to 'abandoned' status after 30 days. Does not include non-basket activities such as browsing products or account management.

indexed — A database optimization where a separate data structure is created to enable fast lookup of records based on the indexed column's values. Indexed columns support efficient WHERE clause filtering, JOIN operations, and sorting, reducing query execution time from linear scans to logarithmic lookups. Indexes incur storage overhead and slow down write operations.

kickoff ui — The user interface component or page where an analyst initiates an analysis run by selecting which requirements or artifacts to analyze, choosing which analysis steps to execute (via checkboxes or similar controls), and configuring analysis parameters. The kickoff UI is the entry point for triggering the analysis workflow and consumes recommended-analyses metadata to pre-select applicable analysis checkboxes.

logged — Recorded in a persistent audit trail or transaction history for compliance, security monitoring, or troubleshooting purposes. In the context of payment transactions, this refers to capturing transaction details, timestamps, and associated metadata in a traceable format. For access control, it refers to recording all attempts to access sensitive member data, creating an audit trail that documents who accessed what information and when.

metadata — A section of an API response containing supplementary information about the result set rather than the data itself, including pagination details (page, page_size, total_count, total_pages), query parameters applied, and response generation timestamp. Metadata is typically returned in a separate JSON object alongside the data array.

migration report — A structured summary document generated by the migration script upon completion (successful or failed) that contains execution metrics including files_processed count, items_created count, duplicates_skipped count, validation_errors count, execution_time_ms, and detailed listings of any duplicate items skipped or validation errors encountered. The report format (JSON, text, log file) and delivery mechanism (console output, file, database record) are implementation-specific.

migration script — A one-time executable program or database script that reads legacy markdown backlog files from the analyst repository, parses their content, validates the extracted data, and inserts backlog items into the backlog_items database table while enforcing transactional semantics and duplicate detection rules. The script is invoked manually by a system administrator with specified application_id and organisation_id parameters, and produces a migration report summarizing execution results.

mission — The overarching purpose and strategic objective of the disruption operations platform: to detect journey irregularities in near real time, orchestrate passenger communications, and execute policy-compliant re-accommodation and compensation workflows with full auditability. It defines the system's core value proposition of translating carrier policies and consumer-rights regimes into deterministic, explainable outcomes for affected passengers.

modal — A modal window or dialog box - a UI overlay that appears on top of the main application content, requiring user interaction before they can return to the underlying page. Modals typically dim or disable the background content, focus user attention on a specific task or message, and are dismissed through explicit user action (clicking a button, pressing ESC, or clicking outside the modal area).

mvp — Minimum Viable Product - a priority classification for backlog items indicating that the item represents core functionality required for the initial product release. MVP items are essential features that must be delivered to meet the minimum threshold for product viability and user value. This is the highest priority classification in the product backlog taxonomy.

orgid — Organisation Identifier - a unique identifier assigned to each organisation (tenant) in the REQQA system, used to scope database queries and enforce tenant isolation. The orgid is derived from the authenticated

user's token and is automatically applied as a filter to all API queries to ensure users can only access data belonging to their organisation.

page — A query parameter specifying which page of results to return in a paginated API response, using 1-based indexing where page=1 returns the first set of results. Used in conjunction with 'page_size' to calculate the offset into the result set (offset = (page - 1) * page_size).

pagination — A technique for dividing large result sets into smaller, manageable pages that can be retrieved incrementally through sequential API requests. Pagination is implemented using limit (maximum number of records per page) and offset (number of records to skip) query parameters, allowing clients to retrieve data in chunks and improving API performance and user experience when dealing with large datasets.

panel — A distinct section or container within a web page user interface that groups related information or functionality, typically with a visible border, heading, and consistent styling. Panels organize content into logical units and may be collapsible, scrollable, or fixed depending on design requirements.

provenance —

rate limiting — A throttling mechanism that restricts the number of API requests a client can make within a specified time window (e.g., 100 requests per minute, 1000 requests per hour) to prevent abuse, ensure fair resource allocation, and protect system stability. Rate limits may be enforced per API token, per user, or per organisation, and exceeded limits result in HTTP 429 Too Many Requests responses with headers indicating when the client can retry.

real-time — A system response characteristic where inventory updates, availability status changes, and stock level modifications are reflected across all system components and user-facing interfaces within milliseconds to low seconds of the triggering event, without perceptible delay from the user's perspective. In the context of inventory management, real-time means that stock decrements from order confirmation, stock reservations from basket additions, and availability status updates are immediately visible to all concurrent users and system processes, preventing race conditions and ensuring data consistency.

reject — To refuse or decline a requested operation, transaction, or state change due to validation failure, business rule violation, or system constraint. The system returns an error code and message explaining the reason for refusal, logs the attempt to the audit trail, and leaves the entity's state unchanged.

requirement template — A reusable structural pattern that defines the format, sections, and default metadata for creating requirements of a specific type (e.g., 29148-full, user story, technical specification). Each template specifies which analyser steps are recommended by default for requirements created from it, and serves as a blueprint ensuring consistency in requirement structure across the organization.

retro-placeholder — A requirement record origin classification (origin='retro-placeholder') indicating that the requirement serves as an anchor point for existing functionality that has not yet been formally documented to current standards. Retro-placeholder requirements have intentionally thin bodies and are excluded from default analyses, reports, and exports until they are retrofitted with complete documentation. They enable tracking of known-existing features awaiting formal specification.

retro-placeholder requirements — Requirements created retrospectively to document functionality that was already implemented without formal specification, serving as placeholders in the requirements model to maintain traceability and completeness. These requirements typically have minimal content, reference existing implementation artifacts, and are marked to indicate their retrospective nature. They are generally excluded from quality analysis runs since they document past decisions rather than specify future work.

retry logic — An automated error-handling mechanism that re-attempts failed email delivery operations after temporary failures (such as network timeouts, rate limiting, or recipient server unavailability). Retry logic includes configurable parameters for maximum retry attempts, backoff intervals between attempts (e.g., exponential backoff), and criteria for distinguishing temporary failures (retryable) from permanent failures (non-retryable).

risks — Potential problems identified during builder's review that could cause implementation difficulties, quality issues, or project delays if not addressed, but do not completely prevent progress. Risks are advisory

findings that should be considered by analysts and may be accepted, mitigated, or resolved before proceeding to design.

scope — A named collection of user stories and their derived requirements that form a cohesive unit of work for development purposes. A scope belongs to exactly one application and serves as the bridge between specification (REQQA) and implementation (Claude Code), capturing what will be built in a particular development effort along with constraints and exclusions.

session — A server-side or client-side state management mechanism that maintains user authentication and context across multiple HTTP requests. A session is created upon successful login, identified by a session token (typically stored in a cookie or local storage), and expires after a defined period of inactivity or when explicitly terminated by logout. The session stores the authenticated user's identity and may cache user preferences to avoid repeated database queries.

sessions — A period of continuous interaction between a user and the system, typically bounded by login/logout events for authenticated users or by browser session lifecycle for guest users. Sessions maintain user state and context across multiple requests, and may expire after a period of inactivity or when explicitly terminated.

severity —

skill — A reusable, configurable capability in Claude Code that encapsulates a specific workflow or operation, invoked via slash commands and defined in .claude/commands/*.md files. Skills can be process skills (platform-agnostic, interacting with external APIs) or platform skills (specific to a development environment).

source — A classification attribute on issues indicating the origin or context in which the issue was identified, with valid values including 'builder_review' (identified during automated analysis) and 'build_phase' (identified during implementation). The source is specified by the API caller during issue creation and is used for reporting and filtering.

step-codes — Short alphanumeric identifiers (such as R-D, R-F, R-C, S-A) that uniquely identify specific analysis types within REQQA. Step-codes are used to reference analysers in metadata, UI selections, and database records, providing a compact notation for analysis types. The 'R-' prefix typically denotes requirement analyses, while 'S-' denotes story analyses.

story — A user story or functional specification unit that describes a discrete piece of functionality from an end-user perspective. Stories are derived from requirements, can be versioned through revisions, and represent the atomic units of work that can be included in or excluded from a scope. Each story belongs to a parent requirement and can appear in multiple scopes simultaneously.

suggestions — Recommendations for improvement identified during builder's review that would enhance requirement quality, clarity, or testability but are not essential for proceeding to implementation. Suggestions are optional enhancements that analysts may choose to incorporate based on time, priority, and value considerations.

system administrator — A user role with elevated privileges responsible for performing system maintenance tasks including database migrations, configuration changes, and operational procedures that affect multiple organizations or the entire platform. System administrators have access to administrative tools and scripts not available to regular users or analysts, and their actions are logged for audit purposes.

template administrator — A user role with permissions to create, modify, and delete requirement templates within their organization, including the ability to configure template structure, default metadata, and recommended analyses. Template administrators manage the reusable patterns that other users employ when creating requirements, ensuring organizational consistency in requirement formats.

tenant isolation — A security mechanism that ensures users can only access data belonging to their own organisation (tenant), preventing cross-organisation data leakage. Implemented by automatically filtering all database queries with the authenticated user's orgid, validating that all referenced entities belong to the same organisation, and rejecting any attempt to access resources from a different tenant with a 403 Forbidden response.

tooltip — A small pop-up text box that appears when a user hovers over or focuses on a UI element, providing additional context, explanation, or help text. Tooltips are typically displayed after a short delay (e.g., 500ms) and disappear when the user moves away from the element. They are used to explain truncated text, provide definitions for icons or abbreviations, or offer guidance without cluttering the main interface.

triage — The process of evaluating and categorizing backlog items to determine their priority, feasibility, and disposition (promote to requirement, decline, or defer). Triage involves analyst review of item details, assessment of business value and technical complexity, and a decision to either promote the item (linking it to a requirement) or decline it (with a documented reason).

type — A classification of tennis balls based on their construction and performance characteristics, with defined values including 'pressurised' (balls with internal air pressure), 'pressureless' (balls without internal pressure), 'training' (designed for practice and coaching), and 'competition' (meeting official tournament standards). Type is a primary filterable attribute in the catalogue.

typical load — The expected normal operating conditions for system performance testing, representing the average concurrent user activity and transaction volume during regular business operations. Typical load is used as a baseline for performance benchmarks and capacity planning, excluding peak periods or stress test scenarios.

update — The action of modifying one or more attributes of an existing investment holding record, including quantity held or cost basis per unit. Updates preserve the holding's identity and history while changing current values, and take effect immediately in portfolio calculations.

version — A numbered or timestamped edition of a plan document that represents a distinct state of the implementation plan at a specific point in time. Each version is immutable once created, with new versions superseding (but not deleting) previous versions, maintaining a complete audit trail of plan evolution. Version numbers typically follow sequential integer numbering (1, 2, 3…) or semantic versioning schemes.

Document provenance

- **Generated:** 01 May 2026 at 15:26
- **Source system:** REQQA (Gerrard Consulting Testing)
- **Scope id:** 274
- **Application id:** 30
- **Notes:** Requested by Paulie G

This document was generated automatically from the REQQA release scope. It reflects the scope's state at the time of generation. Subsequent changes to requirements, stories, or glossary are not included until the document is regenerated.