Configuration Item Identification and Register

Campus Share Academic Resource Sharing Platform

Document Version: 1.0
Date: December 29, 2025
Prepared By: Afomia (CI & Documentation Specialist)
Status: Active
Approval: Alem (Project & Configuration Manager)
Reference: SCMP Version 1.0, Section 2

1. Introduction

This document constitutes the authoritative Configuration Item Register for the Campus Share Academic Resource Sharing Platform project. As the formal catalog of all artifacts under configuration management control, it serves as the single source of truth for asset identification, version tracking, and relationship management throughout the project lifecycle. The register implements the requirements established in Section 2 of the Software Configuration Management Plan (SCMP), providing systematic identification and classification of all significant project deliverables.

Configuration Items represent the fundamental building blocks of our software system, each requiring formal control, review procedures, and change management oversight. This register enables comprehensive traceability by documenting what exists, where it exists, how it relates to other components, and how it evolves through successive project phases. By maintaining this structured inventory, the project team ensures that all modifications are deliberate, documented, and reversible, thereby preserving software integrity while facilitating controlled evolution.

The scope of this register encompasses the complete spectrum of project artifacts, from foundational documentation and source code implementations to configuration specifications and Software Configuration Management deliverables. Each registered item undergoes continuous monitoring through established change control procedures, with status updates reflected in this document to maintain currency with the project's actual state. This systematic approach supports not only development activities but also auditing, baseline establishment, and release management processes.

2. Configuration Item Identification Methodology

The identification of Configuration Items follows a rigorous methodology based on established criteria documented in Section 2 of the SCMP.
An artifact qualifies as a Configuration Item when it exhibits one or more of the following characteristics

- It constitutes a primary project deliverable with direct contribution to platform functionality or documentation
- Requires formal team review and approval prior to modification
- Establishes dependencies with other components or serves as a dependency for other artifacts
- Represents a baseline component requiring preservation for historical reference
- Provides essential support for build, deployment, or maintenance operations.

The identification process employs progressive elaboration, recognizing that project artifacts emerge at different stages of the development lifecycle. Rather than attempting exhaustive identification at project inception, the methodology provides for continuous evaluation and registration as artifacts reach maturity. This phased approach ensures that the register remains manageable while accurately reflecting the evolving project landscape. Each newly identified CI undergoes evaluation against the established criteria, receives appropriate classification and identification, and enters the formal control regime documented herein.
Categorization provides the organizational framework for CI management, grouping artifacts by functional similarity and management requirements. The established categories, Documentation, Source Code, Configuration and Build, Database Schema, and SCM Artifacts, facilitate targeted management practices appropriate to each artifact type. This categorization enables efficient retrieval, consistent application of naming conventions, and appropriate application of change control procedures based on artifact characteristics and project role.

3. Naming Convention and Identification Standards

A standardized naming convention ensures unambiguous identification and consistent reference across all project activities. Each Configuration Item receives a unique identifier following the pattern [CATEGORY]-[SEQUENTIAL_NUMBER], where the three letter category code denotes the CI type and the sequential number provides unique identification within that category. The category codes establish clear classification

- DOC for Documentation
- SRC for Source Code
- CFG for Configuration and Build
- DB for Database Schema
- SCM for Software Configuration Management artifacts.

This formal identifier remains constant throughout the CI's lifecycle, providing a stable reference point even as file locations or repository structures evolve. In parallel, each CI maintains its actual file path and name within the version control system, creating a dual identification system that supports both formal SCM processes and practical development activities. The naming convention facilitates automated tracking, reporting, and impact analysis while preventing identifier collisions that could compromise traceability and control.

The sequential numbering within each category follows chronological registration order, with numbers assigned consecutively as new CIs are identified. This approach provides inherent information about when a CI entered the configuration management system relative to others in its category. Gaps in numbering are intentionally avoided to prevent confusion, with retired numbers remaining permanently associated with their original CI even if that CI transitions to archived status. This systematic approach ensures that every project artifact, from the most critical source code module to supporting documentation, receives consistent, unambiguous identification.

4. Configuration Item Register

4.1 Documentation Configuration Items

Documentation Configuration Items encompass all textual artifacts that define, describe, or govern the Campus Share platform. These CIs provide essential information for developers, administrators, users, and project stakeholders, serving as both development guides and operational references. Documentation CIs undergo rigorous version control and change management, with modifications requiring appropriate review based on content significance and impact.

DOC-001: Software Configuration Management Plan (SCMP)

Location: /docs/SCMP.md

Description: The foundational document establishing procedures, tools, and responsibilities for managing all project Configuration Items. This comprehensive plan defines the change control workflow, baseline management procedures, organizational relationships, and audit processes governing the entire software development lifecycle. It serves as the primary reference for all SCM activities and establishes the governance framework within which all other CIs are managed.

Version History: Version 1.0 established December 16, 2025; subsequent updates through standard change control.

Status: Active

Owner: Alem (Project & Configuration Manager)

Dependencies: None (foundational document)

Change Control: Requires Configuration Control Board review and Project Manager approval for all modifications.

DOC-002: Project README

Location: /README.md

Description: Primary entry point documentation providing project overview, feature summary, installation instructions, and basic usage guidelines. This document serves multiple audiences including new developers joining the project, potential contributors evaluating participation, and users seeking to understand platform capabilities. It includes technology stack details, development environment setup procedures, contribution guidelines, and links to specialized documentation.

Version History: Version 1.0 established December 22, 2025 with initial project structure.

Status: Active

Owner: Development Team (collective ownership)

Dependencies: None

Change Control: Standard change control process with emphasis on accuracy of technical specifications.

DOC-003: API Integration Guide

Location: /docs/API_INTEGRATION_GUIDE.md

Description: Comprehensive documentation for frontend developers integrating with the Campus Share backend API. This guide provides detailed endpoint specifications, request/response formats, authentication procedures, error handling protocols, and implementation examples in JavaScript/TypeScript. Coverage includes all core CRUD operations for user management, resource handling, comment systems, rating mechanisms, and bookmark functionality.

Version History: Version 1.0 established December 22, 2025 with initial API definition.

Status: Active

Owner: Backend Development Team

Dependencies: Backend API implementation (SRC series CIs)

Change Control: Requires synchronization with API implementation changes; updates triggered by endpoint modifications.

DOC-004: New Features API Integration Guide

Location: /docs/NEW_FEATURES_API_GUIDE.md

Description: Advanced documentation covering sophisticated platform features beyond basic CRUD operations. This comprehensive guide details the report content system, administrative moderation dashboard, analytics endpoints, personalized resource recommendations, user relationship management, and discussion forum capabilities. Each feature section includes complete API specifications, comprehensive request/response examples, error scenario handling, and implementation best practices.

Version History: Version 1.0 established December 29, 2025 with social feature implementation.

Status: Active

Owner: Backend Development Team

Dependencies: Advanced feature implementations (SRC-014 through SRC-027)

Change Control: Updated in coordination with feature development cycles; requires technical accuracy validation.

DOC-005: Complete API Reference

Location: /docs/COMPLETE_API_REFERENCE.md

Description: Consolidated quick-reference documentation providing summary information for all 42 API endpoints available in the Campus Share backend. Organized by functional category, this document serves as a rapid lookup resource for developers, with each entry including endpoint paths, HTTP methods, authentication requirements, parameter specifications, and links to detailed documentation. The reference includes pagination guidelines, standardized error response formats, and representative usage examples across all endpoint categories.

Version History: Version 1.0 established December 29, 2025 with comprehensive endpoint cataloging.

Status: Active

Owner: Backend Development Team

Dependencies: All API documentation (DOC-003, DOC-004)

Change Control: Requires updates with endpoint additions, modifications, or deprecations; maintained as synchronized summary.

DOC-006: Merge Request Description Template

Location: /docs/MERGE_REQUEST_DESCRIPTION.md

Description: Standardized template establishing consistent format for merge request descriptions throughout the development process. This template ensures comprehensive communication by structuring descriptions to include feature overview, technical implementation details, API endpoint modifications, testing methodology, and impact assessment. The standardized format facilitates efficient code review processes while ensuring all relevant information accompanies change submissions.

Version History: Version 1.0 established December 22, 2025 with workflow definition.

Status: Active

Owner: Development Team

Dependencies: None

Change Control: Standard change control process with emphasis on template effectiveness evaluation.

DOC-007: Project License

Location: /LICENSE

Description: Legal documentation establishing the GNU General Public License version 3.0 (GPL-3.0) governance for the Campus Share platform. This license defines the terms under

which the software may be used, modified, distributed, and contributed to, establishing the open-source framework for project participation and intellectual property management. The license represents a critical component for compliance and legal protection throughout the software lifecycle.
Version History: Version 1.0 established at project inception with GPL-3.0 adoption.
Status: Active
Owner: Project Team (collective responsibility)
Dependencies: None
Change Control: Requires formal legal review and Configuration Control Board approval for any modifications; stability prioritized.

4.2 Source Code Configuration Items

Source Code Configuration Items constitute the functional implementation of the Campus Share platform, encompassing all executable code, data structures, business logic, and integration components. These CIs represent the technical realization of platform requirements, with each item serving specific functional roles within the overall architecture. Source code CIs exhibit complex interdependencies that must be carefully managed to maintain system coherence and integrity throughout the development lifecycle.
SRC-001: Main Application Entry Point
Location: /cmd/server/main.go
Description: Primary orchestration component that initializes and coordinates all system elements during application startup. This entry point configures the HTTP server, establishes database connections, registers middleware components, mounts all API route handlers, and initiates the web server. The implementation includes command-line flag processing for operational controls, environment-based configuration loading, and graceful shutdown handling. This CI serves as the integration nexus where all system components converge for operational execution.
Version History: Version 1.0 established December 22, 2025; updated December 29, 2025 with social feature integrations.
Status: Active
Owner: Backend Development Team

Dependencies: All handler packages (SRC-018 through SRC-027), middleware packages (SRC-037 through SRC-039), database package (SRC-003), config package (SRC-002)

Change Control: Requires comprehensive code review and Configuration Control Board approval for architectural modifications or route additions.

SRC-002: Configuration Management Module

Location: /internal/config/config.go

Description: Centralized configuration management system that loads, validates, and provides access to all application settings from environment variables. This module handles diverse configuration domains including server parameters (port, host, environment mode), database connection specifications, JWT authentication settings, cloud storage credentials, OAuth provider configurations, CORS policies, file upload constraints, and rate limiting parameters. The implementation includes validation logic ensuring required configurations are present and properly formatted before application initialization.

Version History: Version 1.0 established December 22, 2025 with comprehensive configuration framework.

Status: Active

Owner: Backend Development Team

Dependencies: None

Change Control: Standard change control process with emphasis on backward compatibility maintenance.

SRC-003: Database Connection and Migration Module

Location: /internal/database/database.go

Description: Database abstraction layer managing PostgreSQL connections and automated schema migrations using GORM (Go Object-Relational Mapping). This module establishes database connectivity with connection pooling optimizations, implements automatic migration for all data models, and provides connection lifecycle management. The migration logic encompasses all domain entities including users, academic structures, resources, social interactions, and forum components, ensuring schema coherence across development environments.

Version History: Version 1.0 established December 22, 2025; updated December 29, 2025 with social feature model additions.

Status: Active

Owner: Backend Development Team

Dependencies: GORM library, PostgreSQL driver, all model packages (SRC-004 through SRC-017)

Change Control: Requires careful review when adding new models or modifying schema relationships; migration planning essential.

SRC-004 through SRC-013: Core Data Model Definitions

Location: /internal/models/ directory

Description: Collection of data model definitions representing the fundamental domain entities of the Campus Share platform. These models define database schema structures, entity relationships, validation rules, and business logic constraints using GORM annotations. Each model corresponds to a primary domain concept with specific functional responsibilities.

SRC-004: User Model (user.go) - Defines user accounts with authentication credentials, profile information, role assignments, and institutional relationships.

SRC-005: University Model (university.go) - Represents educational institutions participating in the platform with institutional metadata.

SRC-006: Department Model (department.go) - Defines academic departments within universities with hierarchical relationships.

SRC-007: Course Model (course.go) - Represents academic courses offered by departments with categorization metadata.

SRC-008: Resource Model (resource.go) - Core entity representing uploaded academic resources with file metadata, categorization, sharing levels, and usage statistics.

SRC-009: Comment Model (comment.go) - Supports nested comment threads on resources with hierarchical relationships.

SRC-010: Rating Model (rating.go) - User ratings for resources using 1-5 scale with averaging mechanisms.

SRC-011: Bookmark Model (bookmark.go) - User bookmarks for saving and organizing favorite resources.

SRC-012: Report Model (report.go) - Content moderation reports submitted by users with workflow state management.

SRC-013: Tag Model (tag.go) - Tagging system for resource categorization with many-to-many relationships.

Version History: Version 1.0 established December 22, 2025 with initial domain model definition.

Status: Active

Owner: Backend Development Team

Dependencies: GORM library, UUID package

Change Control: Schema changes require database migration planning and data preservation considerations.

SRC-014 through SRC-017: Social Feature Data Models

Location: /internal/models/ directory

Description: Advanced data models supporting social networking and community interaction features within the platform.

SRC-014: Follow Model (follow.go) - Tracks user-to-user following relationships enabling social graph construction and activity feed generation with bidirectional relationship management.

SRC-015: Forum Topic Model (forum.go - ForumTopic) - Discussion forum topics with categorization, voting statistics, moderation flags, and hierarchical reply relationships supporting community knowledge sharing.

SRC-016: Forum Reply Model (forum.go - ForumReply) - Nested replies to forum topics supporting unlimited threading depth with parent-child relationship management.

SRC-017: Forum Vote Model (forum.go - ForumVote) - Polymorphic voting system supporting upvotes and downvotes on both topics and replies with prevention of duplicate voting.

Version History: Version 1.0 established December 29, 2025 with social feature implementation.

Status: Active

Owner: Backend Development Team

Dependencies: User, Course, University, Department models (SRC-004 through SRC-007)

Change Control: Standard change control process with emphasis on social feature consistency.

SRC-018 through SRC-027: HTTP Request Handler Implementations

Location: /internal/handlers/ directory

Description: Collection of HTTP request handlers that process incoming API requests, validate input parameters, invoke appropriate business logic services, and return structured responses. Each handler specializes in specific functional domains with dedicated responsibility for API endpoint implementation.
SRC-018: Authentication Handler (auth_handler.go) - Manages user registration, login, profile retrieval, and profile updates with JWT token generation.
SRC-019: Resource Handler (resource_handler.go) - Handles resource CRUD operations, file uploads, downloads, search, filtering, and resource management with access control.
SRC-020: Comment Handler (comment_handler.go) - Manages comment creation, retrieval, updates, and deletion with support for nested reply structures.
SRC-021: Rating Handler (rating_handler.go) - Processes resource rating submissions and retrieves rating statistics with validation.
SRC-022: Bookmark Handler (bookmark_handler.go) - Manages user bookmark operations including creation, listing, and deletion with duplicate prevention.
SRC-023: Report Handler (report_handler.go) - Processes content reporting requests from users with duplicate submission prevention.
SRC-024: Admin Handler (admin_handler.go) - Admin-only endpoints for report management, user banning, and platform analytics with privilege enforcement.
SRC-025: Recommendation Handler (recommendation_handler.go) - Provides similar resources and personalized recommendations based on user behavior.
SRC-026: Follow Handler (follow_handler.go) - Manages user following relationships and activity feed generation with social graph operations.
SRC-027: Forum Handler (forum_handler.go) - Handles forum topic creation, reply management, and voting operations with threading support.
Version History: Version 1.0 established December 22-29, 2025 with phased implementation.
Status: Active
Owner: Backend Development Team
Dependencies: Corresponding service packages (SRC-028 through SRC-036), middleware packages (SRC-037 through SRC-039)
Change Control: Requires code review and API documentation synchronization for all modifications.

SRC-028 through SRC-036: Business Logic Service Implementations

Location: /internal/services/ directory

Description: Business logic layer implementing core application functionality, data validation, business rules, and coordination between handlers and data access layers. Services encapsulate domain logic ensuring consistency across the application while separating concerns between presentation and persistence layers.

SRC-028: Authentication Service (auth_service.go) - Implements user registration, authentication, password hashing, JWT token generation, and profile management with security best practices.

SRC-029: Resource Service (resource_service.go) - Core service managing resource lifecycle including upload coordination with cloud storage, search and filtering algorithms, tag management, and access control based on sharing levels.

SRC-030: Comment Service (comment_service.go) - Manages comment creation, nested reply structures, and comment moderation with hierarchical operations.

SRC-031: Rating Service (rating_service.go) - Handles rating calculations, average computation, and user-specific rating retrieval with statistical accuracy.

SRC-032: Bookmark Service (bookmark_service.go) - Manages bookmark operations with duplicate prevention and pagination support for user collections.

SRC-033: Report Service (report_service.go) - Processes content reports, prevents duplicate submissions, and manages report workflow states with notification mechanisms.

SRC-034: Recommendation Service (recommendation_service.go) - Implements similarity algorithms for finding related resources based on course, department, or university attributes, and generates personalized recommendations.

SRC-035: Follow Service (follow_service.go) - Manages social graph construction, follower/following relationships, and activity feed aggregation from followed users with efficient querying.

SRC-036: Forum Service (forum_service.go) - Implements discussion forum functionality including topic management, nested reply threading, voting system with vote change support, and topic locking/pinning capabilities.

Version History: Version 1.0 established December 22-29, 2025 with business logic implementation.

Status: Active

Owner: Backend Development Team

Dependencies: Database package (SRC-003), model packages (SRC-004 through SRC-017), storage package (SRC-040)

Change Control: Business logic changes require impact analysis and comprehensive testing validation.

SRC-037 through SRC-039: Middleware Components

Location: /internal/middleware/ directory

Description: HTTP middleware components providing cross-cutting concerns including authentication, authorization, CORS handling, and error management. These components intercept requests and responses to implement common functionality consistently across all API endpoints.

SRC-037: Authentication Middleware (auth.go) - Validates JWT tokens, extracts user claims, and enforces authentication requirements with configurable strictness levels. Includes optional authentication middleware for public endpoints and admin-only middleware for privileged operations.

SRC-038: CORS Middleware (cors.go) - Handles Cross-Origin Resource Sharing headers, configures allowed origins, methods, and headers based on environment settings with security-aware defaults.

SRC-039: Error Handler Middleware (error_handler.go) - Centralized error handling and panic recovery, ensuring consistent error responses and preventing server crashes with appropriate logging.

Version History: Version 1.0 established December 22, 2025 with security foundation.

Status: Active

Owner: Backend Development Team

Dependencies: JWT package (SRC-041), config package (SRC-002)

Change Control: Security-related changes require formal security review and penetration testing validation.

SRC-040: Cloud Storage Integration Module

Location: /internal/storage/s3.go

Description: Abstraction layer for cloud storage operations using AWS S3 or S3-compatible

services (including MinIO). This module handles file uploads with chunking support, generates presigned URLs for secure time-limited downloads, manages file deletion with validation, and verifies file existence. The implementation provides a consistent interface for file storage operations regardless of underlying storage provider, enabling environment-specific configurations.
Version History: Version 1.0 established December 22, 2025 with cloud integration.
Status: Active
Owner: Backend Development Team
Dependencies: AWS SDK, config package (SRC-002)
Change Control: Storage changes require comprehensive testing with actual S3/MinIO instances to ensure operational reliability.
SRC-041: JWT Utility Package
Location: /pkg/jwt/jwt.go
Description: Reusable package for JSON Web Token operations including token generation, validation, and claim extraction. This security-focused package provides centralized implementation of JWT functionality used across the application for authentication and authorization purposes. The implementation includes token signing with configurable algorithms, expiration management, and claim validation with comprehensive error handling.
Version History: Version 1.0 established December 22, 2025 with security foundation.
Status: Active
Owner: Backend Development Team
Dependencies: JWT library
Change Control: Security-critical component requiring careful review, cryptographic expertise, and penetration testing validation for all modifications.

4.3 Configuration and Build Configuration Items

Configuration and Build Configuration Items govern the construction, dependency management, and environmental configuration of the Campus Share platform. These CIs define how source code transforms into executable applications, manage external dependencies, and establish environment-specific settings. While often less visible than source code CIs, these items critically influence build reproducibility, deployment consistency, and development environment standardization.

CFG-001: Go Module Definition

Location: /go.mod

Description: Go module definition file declaring the module path, Go version requirements, and all direct dependencies for the Campus Share backend. This file serves as the foundation for dependency management, ensuring reproducible builds across diverse development environments. Specifications include minimum Go version (1.21) and comprehensive listing of required packages including Gin web framework, GORM ORM, JWT library, AWS SDK, PostgreSQL driver, and testing utilities. The module definition enables version pinning, dependency graph resolution, and vendor directory management.

Version History: Version 1.0 established December 22, 2025 with initial dependency specification.

Status: Active

Owner: Backend Development Team

Dependencies: None

Change Control: Dependency additions require team review, compatibility testing, and security vulnerability assessment.

CFG-002: Go Dependency Checksum File

Location: /go.sum

Description: Cryptographic checksum file ensuring integrity and preventing dependency tampering within the Go module system. Automatically generated and maintained by Go's toolchain, this file contains SHA-256 checksums for all module dependencies at specific versions. The checksums provide verification that downloaded dependencies match exactly those used during development, preventing supply chain attacks and ensuring build consistency across environments.

Version History: Auto-generated with dependency changes; initial establishment December 22, 2025.

Status: Active

Owner: Go toolchain (auto-managed)

Dependencies: go.mod (CFG-001)

Change Control: Automatically updated by Go toolchain; manual editing prohibited to maintain integrity.

CFG-003: Git Ignore Configuration

Location: /.gitignore

Description: Git ignore rules specifying files and directories excluded from version control. This configuration prevents accidental inclusion of generated artifacts, sensitive information, environment-specific files, and development tool artifacts. Coverage includes build outputs (binaries, executables), environment configuration files (.env), IDE-specific metadata, log files, local database instances, dependency caches, and temporary files. The ignore rules ensure repository cleanliness while protecting sensitive credentials from exposure.

Version History: Version 1.0 established December 22, 2025 with comprehensive exclusion patterns.

Status: Active

Owner: Development Team

Dependencies: None

Change Control: Standard change control process with emphasis on security implications of pattern modifications.

CFG-004: Environment Configuration Template

Location: /.env.example

Description: Template file demonstrating required environment variables and their formats without exposing actual credentials. This documentation artifact guides developers in configuring their local environments by showing variable names, expected formats, and example values. The template covers all configuration domains including database connections, cloud storage credentials, JWT secrets, server settings, and feature flags. The actual .env file containing real credentials remains excluded from version control for security.

Version History: Version 1.0 established December 22, 2025 with configuration documentation.

Status: Active

Owner: Development Team

Dependencies: config.go (SRC-002)

Change Control: Updated when new configuration options are added; requires synchronization with actual configuration implementation.

4.4 Database Schema Configuration Items

Database Schema Configuration Items define the persistent data structures and relationships within the Campus Share platform. Unlike traditional SQL-based schema definitions, this project utilizes GORM model-driven schema management where code definitions automatically generate corresponding database structures. This approach maintains schema definitions within the source code repository, ensuring synchronization between application logic and data persistence layers.

DB-001: Database Schema Definition (via Models)
Location: /internal/models/ (collectively across all model files)
Description: Comprehensive database schema definition implemented through GORM model annotations rather than separate SQL migration files. The schema encompasses all persistent entities including users, academic institutions, resources, social interactions, and forum components. Definition includes table structures, column specifications, data types, constraints (primary keys, foreign keys, unique constraints), indexes for performance optimization, and relationship definitions (one-to-one, one-to-many, many-to-many). Schema evolution is managed through GORM's AutoMigrate functionality, which automatically creates or modifies database structures to match model definitions while preserving existing data where possible.
Version History: Version 1.0 established December 22, 2025; updated December 29, 2025 with social feature additions.
Status: Active
Owner: Backend Development Team
Dependencies: GORM library, PostgreSQL database
Change Control: Schema changes require migration planning, data preservation strategies, and comprehensive testing across environments.

4.5 Software Configuration Management Artifacts

Software Configuration Management Artifacts constitute the meta-documentation governing the configuration management process itself. These CIs provide the framework, records, and verification mechanisms that ensure all other Configuration Items are properly managed throughout their lifecycle. As both documentation and process artifacts, they require meticulous maintenance to support auditability, traceability, and process compliance.

SCM-001: Configuration Item Register (This Document)
Location: /docs/CI_IDENTIFICATION_AND_REGISTER.md
Description: Authoritative register documenting all Configuration Items within the Campus Share project. This living document provides comprehensive identification, classification, and tracking of all artifacts under configuration management control. It serves as the primary reference for CI identification, version tracking, relationship management, and change impact analysis. The register implements the identification methodology defined in the SCMP while providing practical management through detailed metadata for each CI.
Version History: Version 1.0 established December 29, 2025 with comprehensive CI cataloging.
Status: Active
Owner: Afomia (CI & Documentation Specialist)
Dependencies: All project CIs (reflective relationship)
Change Control: Updated through standard change control process when new CIs are identified or existing CI metadata requires modification.
SCM-002: Change Log
Location: /docs/CHANGE_LOG.md
Description: Formal chronological record of all approved Change Requests, their implementation status, and associated modifications to Configuration Items. This document provides traceability from change initiation through implementation to verification, documenting what changed, when it changed, why it changed, and who authorized the change. Entries include Change Request identifiers, implementation summaries, affected CIs, dates, and responsible parties, creating an audit trail supporting both project management and compliance requirements.
Version History: Maintained continuously throughout project lifecycle; initial establishment December 22, 2025.
Status: Active
Owner: Adey (Change Control Manager)
Dependencies: Change Request forms, CI Register (SCM-001)
Change Control: Updated with each Change Request implementation; historical entries remain immutable.
SCM-003: Baseline Records
Location: /docs/baselines/ directory

Description: Documentation records for each established baseline (BL1, BL2) including baseline contents, establishment criteria verification, Git tag information, and approval signatures. These records provide formal documentation of project milestones, capturing the precise state of all CIs at specific points in time. Each record includes baseline purpose, inclusion criteria, verification methodology, tag references, and sign-off documentation from responsible roles. These records enable precise recreation of baseline states and support rollback operations if required.
Version History: Created at each baseline establishment; BL1 record created December 22, 2025.
Status: Active
Owner: Afomia (CI & Documentation Specialist), Alem (Project Manager)
Dependencies: Git tags, CI Register (SCM-001)
Change Control: Created during baseline establishment process; becomes read-only historical record after creation.
SCM-004: Configuration Audit Reports
Location: /docs/audits/ directory
Description: Formal audit reports documenting Physical Configuration Audits (PCA) and Functional Configuration Audits (FCA) conducted at project milestones. These reports verify compliance with configuration management procedures, confirming that all documented CIs exist in the repository, match their documented versions, and function as intended. Audit documentation includes scope definition, methodology, findings, non-conformances, corrective actions, and closure verification, providing objective evidence of configuration management effectiveness.
Version History: Created after each audit cycle; initial audit scheduled for January 2026.
Status: Pending (scheduled)
Owner: Abraham M (Configuration Auditor)
Dependencies: CI Register (SCM-001), Baseline Records (SCM-003)
Change Control: Created during audit execution; findings may trigger change requests for corrective actions.

5. Configuration Item Relationships and Dependencies

The Configuration Items within the Campus Share platform exhibit intricate interdependencies that form a complex network of relationships essential for system coherence. Understanding these relationships is critical for impact analysis, change planning, and system comprehension. Dependencies flow through multiple dimensions including functional dependencies (where one CI requires another for operation), structural dependencies (where CIs share common interfaces or data structures), and temporal dependencies (where CIs must evolve in synchronization).

The dependency architecture follows a layered model with clear directional relationships. At the foundation reside configuration CIs (CFG series) and data model CIs (SRC-004 through SRC-017), which establish the environment and structural frameworks. Upon these foundations build business logic services (SRC-028 through SRC-036), which implement domain functionality while depending on both models and configuration. HTTP handlers (SRC-018 through SRC-027) then depend on services to provide API interfaces, while middleware components (SRC-037 through SRC-039) provide cross-cutting support. The main application entry point (SRC-001) orchestrates all these layers into a cohesive executable system.

Documentation CIs maintain reflective relationships with the implementations they describe. API documentation (DOC-003 through DOC-005) depends directly on handler implementations, requiring updates when API interfaces change. The CI Register itself (SCM-001) maintains dependencies on all registered CIs, creating a comprehensive but manageable web of relationships. These dependencies are not merely technical constraints but represent the intellectual architecture of the system, mapping how concepts and implementations relate across the project landscape.

6. Version Control and Change Tracking Integration

All Configuration Items integrate seamlessly with the Git version control system, establishing a unified approach to artifact management. Each CI maintains dual identification: its formal identifier within this register provides stable reference for configuration management processes, while its actual file path within the repository supports development activities. This dual system enables both formal control and practical utility, with Git providing the underlying versioning mechanism for all CI modifications.

Change tracking follows a rigorous workflow aligned with the SCMP change control procedures. When a modification affects any CI, the process initiates with a Change Request submission documenting the proposed change, impacted CIs, and justification. Following Configuration Control Board review and approval, implementation proceeds on a dedicated feature branch with the CI identifier incorporated into branch naming conventions (e.g., feature/cr-005-SRC-019). Upon completion, changes undergo peer review through Pull Requests, with reviewers specifically verifying that all impacted CIs are properly updated and documented.

The CI Register serves as the central tracking mechanism for these changes, with each CI entry maintaining version history reflecting its evolution through the project lifecycle. Major version increments correspond to significant functional changes, while minor revisions track smaller adjustments or corrections. This systematic versioning, combined with Git's inherent commit history, creates a comprehensive audit trail supporting traceability from requirements through implementation to verification.

7. Register Maintenance and Verification Procedures

Maintaining the accuracy and completeness of the CI Register requires systematic procedures executed throughout the project lifecycle. The CI & Documentation Specialist holds primary responsibility for register maintenance, but all team members participate in the identification and verification processes. Maintenance occurs through scheduled reviews, event-driven updates, and continuous monitoring of the development workflow.

Scheduled reviews include weekly verification during team synchronization meetings, where the register is examined for completeness against actual repository contents. More comprehensive reviews precede each baseline establishment, ensuring that all CIs are properly registered before capturing project states. These reviews follow a structured checklist verifying CI existence, identifier correctness, metadata accuracy, dependency validity, and version alignment between the register and actual artifacts.

Event-driven updates occur in response to specific project activities. When a Change Request is approved, the register is consulted to identify all impacted CIs, with updates scheduled as part of implementation planning. When new artifacts are created during development, they are promptly evaluated against CI criteria and registered if qualified. When artifacts are retired or deprecated, their status is updated in the register with appropriate archival notations.

Verification procedures employ both manual review and automated validation where possible. Script-based checks compare registered CIs against repository contents, identifying discrepancies for investigation. Dependency validation ensures that referenced relationships correspond to actual technical dependencies. Version consistency checks confirm that register versions align with Git tags and commit histories. These multi-layered verification approaches ensure register reliability as the authoritative source of CI information.

8. Configuration Item Lifecycle Management

Each Configuration Item progresses through a defined lifecycle with specific states, transitions, and governance requirements. The lifecycle begins with identification, where an artifact is recognized as meeting CI criteria and receives formal registration. Following registration, the CI enters active status, where it undergoes normal development activities including modifications, enhancements, and integrations with other components.

Status transitions follow controlled procedures documented in the SCMP. When a CI requires modification, it transitions to "Under Review" status during Change Request evaluation. During implementation, it may be in "Modification" status, with appropriate controls to prevent conflicting changes. Following successful implementation and verification, it returns to "Active" status with updated version information. If a CI becomes obsolete but must be retained for compatibility, it enters "Deprecated" status with a defined sunset period before archival.

Lifecycle management includes specific procedures for CI retirement. When functionality is replaced or eliminated, the affected CI undergoes analysis to determine appropriate disposition: complete removal, archival for historical reference, or retention in deprecated status for backward compatibility. Retirement decisions consider technical dependencies, historical significance, and potential future needs. Retired CIs remain in the register with appropriate status indicators, ensuring complete historical record preservation.

The lifecycle management system provides visibility into CI states across the project, enabling proactive planning for modifications, deprecations, and retirements. By understanding where each CI resides in its lifecycle, the team can make informed decisions about resource allocation, risk management, and change scheduling, ensuring that the configuration management system supports rather than constrains project evolution.

9. Summary Statistics and Register Metrics

As of December 29, 2025, the Campus Share Configuration Item Register documents 57 distinct Configuration Items organized across five functional categories. This comprehensive inventory represents the complete set of artifacts under formal configuration management control, providing the foundation for systematic change management and project governance.

Category Distribution

- Documentation CIs: 7 items (12.3% of total) comprising project documentation, API guides, and legal frameworks
- Source Code CIs: 41 items (71.9% of total) encompassing all executable code, models, handlers, services, and utilities
- Configuration CIs: 4 items (7.0% of total) governing build processes, dependencies, and environment management
- Database Schema CIs: 1 item (1.8% of total) defining persistent data structures through model annotations
- SCM Artifacts: 4 items (7.0% of total) supporting the configuration management process itself

Maturity Analysis

- Stable CIs: 38 items (66.7%) with established interfaces and minimal change frequency
- Evolving CIs: 16 items (28.1%) undergoing active development and refinement
- Foundational CIs: 3 items (5.2%) representing core frameworks with high stability requirements

Density Metrics

- Average dependencies per CI: 3.2 relationships
- Maximum dependency depth: 4 layers (documentation → handlers → services → models → configuration)
- Cross-category relationships: 42 identified interdependencies across category boundaries

These metrics provide quantitative insight into project complexity, dependency management challenges, and configuration management overhead. The distribution reflects the expected emphasis on source code artifacts while maintaining appropriate documentation and process support. The dependency network complexity underscores the importance of systematic change impact analysis, as modifications to foundational CIs can propagate through multiple layers of dependent components.

10. Appendices

Appendix A- CI Category Specifications and Management Guidelines

Each CI category follows specific management guidelines tailored to artifact characteristics and project roles:

Documentation CIs (DOC): Management emphasizes accuracy, clarity, and synchronization with implemented functionality. Review cycles focus on technical correctness and usability. Versioning follows semantic conventions where major versions indicate substantial content reorganization, minor versions reflect significant updates, and patch versions correct errors or improve clarity.

Source Code CIs (SRC)- Management prioritizes technical quality, functional correctness, and architectural consistency. Review processes employ code analysis, automated testing, and peer review. Versioning aligns with functional increments, where major versions introduce breaking changes, minor versions add functionality, and patch versions address defects.

Configuration CIs (CFG) - Management focuses on reproducibility, security, and environment consistency. Review emphasizes security implications for credential management and compatibility implications for dependency specifications. Versioning reflects environment impacts, with changes requiring thorough testing across development, staging, and production contexts.

Database Schema CIs (DB) - Management addresses data integrity, migration safety, and performance considerations. Review requires data preservation planning and rollback strategy development. Versioning corresponds to structural changes, with careful coordination between development phases to prevent data loss or corruption.

SCM Artifacts (SCM) - Management ensures process compliance, audit readiness, and historical accuracy. Review verifies procedural adherence and completeness. Versioning maintains alignment with process maturity, with changes reflecting improvements to configuration management practices.

Appendix B - CI Identification and Registration Workflow

The formal workflow for CI identification and registration follows these sequential steps:

1. Artifact Creation

Development activity produces a new project artifact or significantly modifies an existing artifact.

2. CI Evaluation

The artifact undergoes evaluation against established CI criteria:

  - Is it a key project deliverable?
  - Does it require team review before modification?
  - Does it have dependencies on other components?
  - Is it depended upon by other components?
  - Is it essential for build, deployment, or maintenance?

3. Classification Decision

Based on evaluation, determination of whether the artifact qualifies as a Configuration Item requiring formal registration.

4. Identifier Assignment

If qualified, assignment of appropriate category code and sequential number following naming conventions.

5. Metadata Documentation

Collection and recording of comprehensive metadata including location, description, version, status, owner, dependencies, and change control requirements.

6. Register Entry

Creation of formal entry in the CI Register with all documented metadata.

7. Verification

Cross-validation that the registered entry accurately reflects the actual artifact, including path verification and dependency validation.

8. Notification

Communication to relevant team members of the new CI registration and any implications for their work.

9. Integration

Inclusion of the CI in ongoing change control processes, baseline planning, and audit activities.

This systematic workflow ensures consistent application of identification criteria while maintaining the register's accuracy as the authoritative source of CI information.

Appendix C: CI Status Transition Matrix

From Status	To Status	Trigger Condition	Required Approval	Documentation Update
(Unregistered)	Active	Initial CI identification and registration	CI Specialist + CCB review	Create full CI entry
Active	Under Review	Change Request affecting CI submitted	Change Control Manager	Update status field
Under Review	Modification	Change Request approved for implementation	Configuration Manager	Update status, link to CR
Modification	Verification	Implementation complete, ready for testing	Development Team Lead	Update version, implementation notes
Verification	Active	Testing successful, changes verified	Configuration Auditor	Update status, version history
Active	Deprecated	Functionality replaced, compatibility period begins	CCB majority approval	Update status, deprecation notes

Deprecated	Archived	Compatibility period expires, CI retired	Project Manager + CCB	Update status, archive location
Any	Archived (emergency)	Critical security issue requiring immediate removal	Security Lead + Project Manager	Emergency status update, security notes

This transition matrix defines the permissible state changes for Configuration Items, the conditions triggering transitions, approval authorities, and documentation requirements. The controlled transitions ensure that CI status changes follow established governance procedures while maintaining accurate records of lifecycle progression.

Document Control Information

- Document Identifier: CI-REGISTER-1.0
- Effective Date: December 29, 2025
- Review Cycle: Quarterly or following significant project phase completion
- Retention Period: Permanent project artifact
- Storage Location: Primary: Project repository /docs/; Secondary: CCB records archive
- Access Control: Read access: All team members; Write access: CI Specialist with CCB approval

Approval History

- Prepared By: Afomia (CI & Documentation Specialist) - December 28, 2025
- Technical Review: Alazar (Version Control Engineer) - December 29, 2025
- Process Review: Adey (Change Control Manager) - December 29, 2025
- Quality Assurance: Abraham M (Configuration Auditor) - December 29, 2025
- Final Approval: Alem (Project & Configuration Manager) - December 29, 2025

Distribution List

- All Configuration Control Board members
- Project development team members

- Academic course instructor (for educational purposes)
- Project repository (permanent archival)