PawfectMatch

CONFIGURATION MANAGEMENT PLAN

Version 1.1

28/10/2025

VERSION HISTORY

Version #	Implemented By	Revision Date	Approved By	Approval Date	Reason
1.0	Chong Yao	20/10/2025	Zi Jian	25/10/25	Configuration Management Plan Template & initial draft
1.1	Chloie	28/10/2025	Zi Jian	28/10/2025	Configuration Management Plan Finalization

TABLE OF CONTENTS

1 IDENTIFICATION

1.1 DOCUMENT OVERVIEW

This document defines the Configuration Management (CM) strategy for the *PawfectMatch* project, a web-based platform connecting pet owners with service providers such as groomers, walkers, and sitters.

The plan outlines the processes and responsibilities for identifying, controlling, tracking, and auditing configuration items (CIs) to maintain integrity, consistency, and traceability across all project deliverables.

This plan applies to all phases of the project lifecycle and is intended for use by the entire project team , particularly the Project Manager, Release Engineer, and QA Manager, who are jointly responsible for its execution.

1.2 ABBREVIATIONS AND GLOSSARY

The following subsections define the standard terminology, abbreviations, and definitions used throughout this Configuration Management Plan. Establishing this common vocabulary is essential for ensuring clear, unambiguous communication among all team members and stakeholders.

1.2.1 Abbreviations

Abbreviation	Full Form
CI	Configuration Item
CM	Configuration Management
QA	Quality Assurance
PM	Project Manager
ITS	Issue Tracking System (JIRA, GitHub Issues)
RE	Release Engineer
SSoT	Single Source of Truth

1.2.2 Glossary

Term	Definition
Configuration Item (CI)	Any component under version control — source code, documents, test plans, or deployment scripts
Baseline	A formally approved version of a product or document that serves as a starting point for further development
Version	A specific state of a configuration item identified by a unique number or tag in GitHub
Reference Configuration	The set of CIs that constitute a defined project milestone (e.g., after system test or release)
Change Request (CR)	A formal proposal to modify a configuration item or baseline
Single Source of Truth (SSoT)	A central, protected repository (GitHub) that holds the one and only official version of all CIs

2 ORGANIZATION

2.1 ACTIVITIES AND RESPONSIBILITIES

The responsibilities for configuration management are distributed among three key roles, the Release Engineer (RE), the Project Manager (PM), and the Quality Assurance (QA) Manager.

Setting Up the Project

Activities	Person Responsible
Define the CM processes and identification rules	Release Engineer (RE)
Install and set up the software configuration repository	Release Engineer (RE)
Set up the Issue Tracking System (JIRA/GitHub Issues)	Release Engineer (RE)
Define and manage the repository structure and access controls	Release Engineer (RE)

Project Lifecycle

Activities	Person Responsible
Manage and control all Configuration Items (CIs)	Release Engineer (RE)
Oversee the CI/CD pipeline and automated checks	Release Engineer (RE)
Create and tag all official versions and releases	Release Engineer (RE)
Manage and archive configuration records and documentation	Release Engineer (RE)
Manage repository backups	Release Engineer (RE)
Verify and approve all reference configurations and baselines	Project Manager (PM)
Authorize the final delivery/deployment of a release	Project Manager (PM)
Conduct formal configuration audits at each milestone	QA Manager
Inspect all configuration records for compliance	QA Manager

Track bug reports and manage quality-related records	QA Manager

2.1.1 Decisions process and responsibilities

All configuration decisions, reviews, and audits are formalized to ensure traceability and accountability. The process and responsibilities are divided into two main scenarios, baseline creation and formal audits.

Baseline Approval

This process done at the end of a sprint or major milestone and is used to freeze a set of CIs and establish a new formal baseline

Activities	Person Responsible
Propose and execute a configuration freeze on the repository	Release Engineer (RE)
Present the final configuration state of all CIs	Release Engineer (RE)
Present the final documentation state	Release Engineer (RE)
Review the baseline for compliance with quality standards	QA Manager
Provide final approval for the new baseline	Project Manager (PM)

Configuration Management Process Audit

This is a formal audit to ensure the CM process is being followed correctly.

Activities	Person Responsible
Initiate and conduct the formal process audit	Project Manager (PM)
Present all configuration state records and logs	Release Engineer (RE)
Present all quality records, bug reports, and test results	QA Manager
Present all documentation management records	Release Engineer (RE)

3 CONFIGURATION MANAGEMENT

3.1 IDENTIFICATION RULES

3.1.1 Identification Rules for Configuration Items

All project artifacts are classified as Configuration Items (CIs) and are identified using a standardized naming convention and versioning system

3.1.1.1 List of Configuration Items

The following artifacts are classified as CIs and are subject to the controls in this plan.

- **Source Code:** All frontend and backend application code
- **Database:** PostgreSQL database schema and all versioned migration scripts
- **Automation Scripts:** All CI/CD pipeline scripts
- **Core Documentation:** All formal project documents
- **QA Artifacts:** All formal test plans and test reports

3.1.1.2 Versioning Standard

While individual files follow developer-chosen names, all official releases and baselines are versioned using semantic versioning. This is the official versioning system for this project.

- **Format:** MAJOR.MINOR.PATCH (e.g. v1.2.3)
- **MAJOR:** Incremented for incompatible API changes or a new baseline release
- **MINOR:** Incremented for new, backward-compatible feature additions
- **PATCH:** Incremented for backward-compatible bug fixes

3.1.2 Identification Rules of Documents

All formal project documents are CIs and are managed with a two-part identification system, location and internal versioning.

3.1.2.1 Identification by Location

All formal documents, such as the Project Plan, SRS and this CMP, are stored in the central /Documentation folder within our GitHub repository. This provides a single, controlled location for all non-code artifacts.

3.1.2.2 Identification by Version History

While filenames are descriptive, the official version of a document is defined internally. Every formal document is required to contain a Version History table.

This table records the evolution of the document and includes:

- Incremental version number
- Author
- Date of Revision
- Approver
- Reason for change

This internal version block, combined with GitHub's commit history, provides a complete and auditable trail for all document modifications.

3.2 REFERENCE CONFIGURATION IDENTIFICATION

A reference configuration is the specific set of CI versions that constitutes a formal, approved Baseline. Each reference configuration is defined by three components:

1. **Unique Identifier:** Semantic version tag applied in GitHub repository to create an immutable snapshot of all CIs in the reference configuration
2. **Descriptive Content:** Contents of the configuration are listed in the GitHub Release Notes associated with that version tag, which are derived from our formal Release Plan
3. **Formal Validation:** Configuration must be associated with a formal approval event, such as a User Acceptance Testing (UAT) sign-off or a QA audit

A reference configuration and its corresponding baseline are established for each major project milestone, including (but not limited to):

- Prototype Demo Baseline
- Final Submission Baseline
- Any other major MINOR or MAJOR version release

3.3 CONFIGURATION BASELINE MANAGEMENT

A baseline is a formally approved snapshot of one or more CIs that serves as an official reference point for all future development and change. We will establish three main types of baselines throughout the project lifecycle.

Types of Baselines

- **Functional Baseline (FBL):** Describes the system's functional and non-functional requirements
- **Allocated Baseline (ABL):** Describes the high-level and detailed design of the system
- **Product Baseline (PBL):** Consists of completed, tested, and accepted system components and their corresponding documentation
- **Development Baseline:** Indicates the state of work products amid development

Baseline Control

A new baseline is formally established only after it has met all three of the following control gates:

1. Release Engineer has created the reference configuration and confirmed its completeness
2. QA Manager has conducted a formal Configuration Audit and verified that the baseline meets all quality standards
3. Project Manager has provided the final approval for the baseline

Once approved, a baseline is frozen and can only be changed through the formal Change Management process.

4 CONFIGURATION CONTROL

Configuration control is a method for controlling, approving, and tracking changes to a product's deliverables and processes.

4.1 CHANGE MANAGEMENT

All changes to an established, approved baseline are governed by the following formal change management workflow. This process ensures that every modification is documented, tested, reviewed, and traceable.

1. **Change Initiation**
    - Any stakeholder identifies a bug or a required enhancement
    - A formal Change Request (CR) is created as a ticket in our Issue Tracking System (ITS), detailing the change, its justification, and its priority
2. **Change Approval**
    - The Project Manager (PM) reviews the CR
    - If approved, the PM assigns the ticket to a developer to begin work
3. **Branching and Development**
    - The developer creates a new, isolated feature branch from the main branch, using the ITS ticket identifier in the branch name
    - The developer implements the required changes in this branch and all commits must reference the CR ticket number for traceability
4. **Quality Assurance**
    - When work is complete, the developer opens a Pull Request (PR) to merge their branch into main
    - This action automatically triggers our CI/CS pipeline, which runs all mandatory automated checks
        - Code formatting to enforce consistent style
        - Linting to detect errors and enforce coding standards
        - Static type checking to ensure type correctness
        - Unit tests to verify individual components function as expected

- Simultaneously, the PR is assigned to another team member for a mandatory Peer Review

5. **Change Integration:** A change is only allowed to be merged into the main branch after it has passed both quality gates
6. **Closure:** Once merged, the original branch is deleted, and the corresponding ticket in the ITS is marked as resolved, formally closing the change loop

The same process is applied for multiple configurations (parallel change/versions), with the distinction being the number of concurrent changes being managed.

5 CONFIGURATION SUPPORT ACTIVITIES

5.1 CONFIGURATION STATUS ACCOUNTING

Configuration Status Accounting (CSA) is the process to record, store, maintain and report the status of configuration items during the software lifecycle. All software and related documentation should be tracked throughout the software life. It ensures that all configuration data and documentation are recorded as each CI progresses through its life cycle from test to production to retirement.

5.1.1 Evolutions traceability

The traceability of all modifications is maintained by linking our ITS, version control, and release tools. This creates a clear, auditable trail from a requirement to the code that implements it.

- **Document:** Traceability maintained via the internal Version History table in each document and the Git commit history for the /Documentation folder
- **Source Code:** Traceability is enforced by our Change Management workflow
    - **Origin:** The change originated from a ticket in our ITS
    - **Implementation:** The code is written in a branch named after the ticket
    - **Modification:** The Git commit messages are required to reference the ticket number
    - **Integration:** The Pull Request links the branch and commits back to the original ITS ticket
- **Releases:** The GitHub Release Notes for a version provide a summary of all changes, features, and bug fixes that are included in that new baseline

5.1.2 Configuration Status Reporting

The Project Manager and Release Engineer maintain status dashboards in JIRA showing the state of all features and CIs.

5.1.3 Configuration Status Diffusion

Updates on baseline changes and CI statuses are communicated during weekly team stand-ups and via GitHub notifications.

5.1.4 Configuration Status Records Storage

All records (issues, pull requests, releases, audit reports) are stored in GitHub and archived in Google Drive.

Records are retained for one year after project closure.

5.2 CONFIGURATION AUDITS

Configuration audits are conducted to ensure:

- All approved changes are implemented correctly.
- System builds match the approved baseline.
- Documentation accurately reflects the software.

Types of audits:

- **Functional Audit:** Verifies functionality meets requirements.
- **Physical Audit:** Verifies CIs exist and are correctly identified.
- **Process Audit:** Reviews compliance with configuration procedures.

Audits are performed by the QA Manager at each major release milestone, with results documented and approved by the Project Manager.