Repository Protection Policy Document

Version 1.0.1

Persons In Charge

Name	Team
Koay Wei Teng	Quality Assurance (Version control)
Cheong Jin Xuan	Quality Assurance (Version control)
Chia Pei Xin	Quality Assurance (Version control)
Ng Jun Bin	Quality Assurance (Version control)
Eng Jun Cheng	Quality Assurance (Version control)
Teh Kuan Kiat	Quality Assurance (Version control)

Document History

Date	Version	Document Revision Description	Document Author
17/4/2024	1.0.0	● Establish the main framework of Repository Protection Policy Documentation ● Document the rational introduction behind implementing a Repository Protection Policy to safeguard the organisational source code and data ● Outline key GitHub concepts such as organisation and repositories to provide a foundational understanding for readers unfamiliar with GitHub ● Define the primary repository protection policies, including guidelines for branch management protection rules and the access control, commit message rule and best practices, pull request rules,	Cheong Jin Xuan

		code review practices etc ● Specify standard operations procedures for handling pull requests, merging code changes, for Quality Assurance Team members to comply with the defined standards	
23/4/2024	1.0.1	● Include instructions detailing how individual developers can merge their changes into the main branch. ● Rename the primary branch from "main / master" to "main" ● Correct the typo of “Github” to “GitHub” ● Add and include APA-formatted references in Section 4.0 References ● Enhance Section 2.0 GitHub main concepts by providing more details and information of roles in an organisation, commits, pull requests, and issues concepts.	Cheong Jin Xuan

Approvals

Approved Date	Approved Version	Approver role	Approver

Table of Contents

1.0 Introduction

1.1 Overview

The Repository Protection Policy Document outlines the guidelines and procedures for safeguarding and managing the project coding assets including all the source code and the relevant repositories hosted using a version controlled system within an organisation. This document aims to serve the purpose of ensuring the security, confidentiality and availability of project code related information stored in the repositories. It defines the roles and responsibilities of protection policy and also establishes the access controls measures within an organisation to secure all the source code. The policy also addresses compliance with relevant regulations and standards, such as data protection laws and industry-specific requirements. By implementing this policy, the organisations can reduce risks associated with information breach and loss, unauthorised access to the organisation assets and maintain the trust among the members of the organisation.

1.2 Version controlled platform

The version controlled platform chosen is GitHub. GitHub is a widely used platform for version control and collaboration on software development projects which provides tools for centralised storage, code management, code review and code changes tracking. GitHub has widespread adoption and many comprehensive features tailored for software development. The code review capabilities enable efficient code reviews by the quality assurance team, enhancing code quality by maintaining a clean code architecture and facilitates the collaboration among all team members. Additionally, GitHub's issue tracking system provides a collaborative environment for managing tasks, bugs, and feature requests. With its strong community support and extensive integration options GitHub will streamline the project development workflows and assist in delivering high-quality software products efficiently.

2.0 GitHub main concepts

2.1 Organisation

Organisations on GitHub serve as collaborative spaces where the software projects and its relevant teams can work together with advanced security and administrative features. In this project, an organisation will be created by the Quality Assurance - Version Control Team and all the team members will be invited to join the organisation by the Quality Assurance - Version Control Team. Each member that joins the organisation can sign into their personal account to collaborate on shared projects. The organisation owners have the authority to manage access controls and permissions for the organisation's resources, including repositories, projects, and apps. All of the Version Control team members will be assigned as organisation owners.

2.2 Roles in an organisation

GitHub organisations offer a flexible role-based access control system that allows organisation owners to assign specific roles to individuals and teams. This is to determine their level of access and permissions within the organisation. The roles can range from organisation owners with complete administrative control to members with basic repository access. For example, there are specialised roles such as moderators, billing managers, security managers, GitHub App managers, and outside collaborators for different versions of GitHub platform. Each role comes with a predefined set of permissions, such as creating repositories, managing billing information, inviting or removing members, and more.

Roles	Description
Organisation owners	Have complete administrative access and control over the organisation.
Organisation members	Default role for individuals with basic permissions to create repositories and projects.
Billing managers	Users who can manage billing settings, including payment information, for the organisation.
Security managers	Roles that grant permissions to view security alerts and manage code security settings across the organisation.
GitHub App managers	Individuals authorised to manage the settings of GitHub App registrations owned by the organisation.
Outside collaborators	External individuals with access to specific organisation repositories without being organisation members.

2.3 Repository

After getting the feedback from front-end and back-end teams and further discussion with all of the other groups, a single repository combining Web application, Desktop application, and backend services components will be created based on the monolithic architecture. This consolidates all related project files and codebases into one centralised repository. This approach has benefits of streamlining version control, facilitating the collaboration of across different team members and enhances the overall project management. Developers from different teams will work on different aspects of the application, such as Web frontend user interfaces, Desktop frontend user interfaces, backend services algorithm logic etc. This facilitates easier code referencing and reduces the overhead associated with managing multiple repositories.

2.4 Commits

A commit captures the changes made to files in your branch by creating a snapshot of the modifications in a repository. Git assigns a unique identifier, known as a SHA or hash, to each commit, which tracks the specific changes, the time they were made, and the author responsible. When creating a commit, it's essential for the authors to provide a brief description of the changes in a commit message. Additionally, the authors also have the option to include a co-author for commits that involve collaboration.

2.5 Branches

Branches on GitHub allow developers to work on new features, bug fixing issues, or test out ideas without impacting the main repository. Typically, a new branch is created from an existing one, usually from the main branch. This setup enables the developers to work separately from ongoing changes in the repository. These branches which are designed for particular features or subjects, are often referred to as feature branches. An individual developer must have write access to a repository to create a branch, open a pull request, or delete and restore branches in a pull request.

2.6 Pull Requests

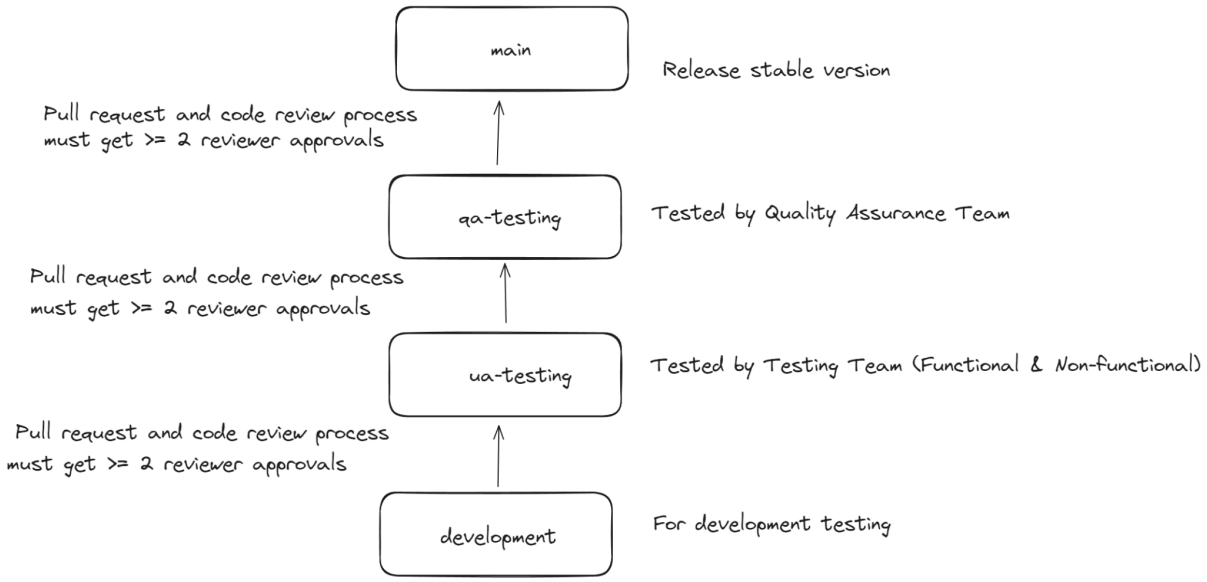
Pull requests allow users to propose changes made to a branch in a repository. It will facilitate collaboration and discussion among collaborators before merging the changes into the targeted branch. Pull requests display the differences between the source and target branches. Hence, the team members can review, discuss, and make additional notes within the pull request for advice. Pull requests can be created and managed through various GitHub interfaces. After initiating a pull request, team members can review changes, add labels, milestones, and assignees, and mention other contributors for feedback. Once satisfied, changes can be merged into the targeted branch, subject to passing required status checks.

2.7 Issues

GitHub Issues serve as a versatile tool for tracking ideas, feedback, tasks, or bugs within a GitHub repository. By creating issues to track progress, it helps the team members to facilitate exchanging feedback, collaborating on ideas, and communicating effectively with each other. Integrated with GitHub's ecosystem, issues enable seamless tracking of related work. When an issue is referenced in another issue or pull request, its timeline updates to reflect the cross-reference. Additionally, we can link an issue to a pull request to represent working progress on a specific ongoing work, and once the pull request is merged, the associated issue will automatically close.

3.0 Repository protection policy guidelines

3.1 Important Branches



No	Branch name	Description
1	development	The development branch serves as the primary branch where new features and new code changes are developed. This branch allows developers to conduct experiments, local testing on new functionalities to make sure no changes are affecting the main codebase. Once the changes are complete and tested, developers can open a pull request to merge the code into the ua-testing branch.
2	ua-testing	The ua-testing branch is used for integrating and testing the codebase to ensure that all features and the system are working together as expected. After pull requests to the development branch are approved, the changes into the development branch will be merged into the ua-testing branch by a pull request. They are integrated into this branch for comprehensive system testing by the Testing Team. At least two reviewers (**1 from Functional Testing team & another 1 from Non-Functional Testing team**) must approve a pull request before it can be merged into the ua-testing branch.
3	qa-testing	The qa-testing branch is dedicated to quality assurance testing, where all the quality assurances will validate the software's functionality against business requirements. Once the code changes pass ua-testing conducted by the Testing Team in the ua-testing branch, pull requests will be opened into the qa-testing branch for final testing validation by the Quality Assurance Team. At least two reviewers **(1 from QA Standard & Compliance & another 1 from QA Version Control)** must

		approve a pull request before it can be merged into the qa-testing branch.
4	main	The main / master branch in a Git repository represents the stable version of the project codebase which is ready to be released. It will hold the latest release that has undergone comprehensive testing and quality checks in previous branches. To ensure the software meets the quality standards. Once all the testing in the previous branches are approved, pull requests will be opened to the main / master branch to merge the changes into this branch. At least two reviewers **(done by QA teams)** must approve a pull request before it can be merged into the main / master branch.

3.2 Collaborator permission

No	Permission	Description
1	Write permission	By default all project team members are allowed to write (Members will be able to clone, pull and push all repositories).

3.3 Branch protection rules (Applied on main & qa-testing & ua-testing)

No	Rule	Description
1	Require pull request reviews before merging	This setting ensures that no pull request can be merged until it has been reviewed by the Quality Assurance Team. Code reviews provide an opportunity for team members to assess code changes before they are integrated into the branches.
2	Require 2 approvals before merging	At least two (2) reviewers must approve a pull request before it can be merged into all branches except the development branch only. This helps maintain code quality and reduces the risk of introducing errors or bugs into the targeted branches.
3	Require GitHub status checks to pass before merging	Status checks of GitHub must pass successfully and checked by the Quality Assurance Team before a pull request can be merged.
4	Require development branches to be up to date before merging	Before merging a pull request into a target branch (e.g., main / qa-testing / ua-testing), the pull request must be based on the latest changes in that branch. This prevents conflicts and ensures that changes are applied to the most current version of the codebase.
5	Dismiss stale pull request approvals when new commits are pushed	The stale pull request approvals that have new commits pushed will be discarded to clear outdated approvals in pull request reviews by the Quality Assurance Team. As new commits are pushed, the previously given approvals can become irrelevant due to changes in the codebase.

6	Disallow force pushes	All developers are prohibited from the use of force-push commands (git push --force) to overwrite the history of the protected branch. This prevents accidental or malicious deletion of commits.
7	Disallow branch deletion	All developers are prohibited from the deletion of the protected branch which could lead to loss of important commit history and changes.
8	Enforce all configured restrictions applied to all members, including administrators	Quality Assurance Team should ensure that even repository administrators are subject to the same restriction as other members when making changes to the protected branch.
9	Require approval of the most recent reviewable push, the most recent reviewable push must be approved by someone other than the person who pushed it	The most recent set of changes pushed to the targeted branches must be approved by at least one reviewer who did not contribute those changes. This ensures that contributors don't approve their own code changes.

3.3 Commit message rule

Developers are advised to use the following types in their commit messages in order to maintain best practices and clean code history.

No	Type	Description
1	feat	New feature
2	fix	Bug fixing
3	docs	Documentation updates
4	style	Code style/formatting changes
5	refactor	Code refactoring
6	test	Adding or modifying tests
7	chore	Routine tasks, maintenance, or tooling changes

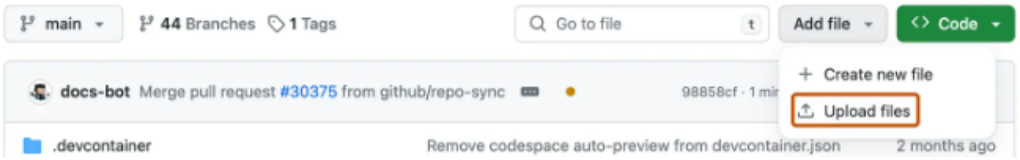
Example of commit message:

1) feat: add OCR model for handwritten digit recognition

2)

Notes:

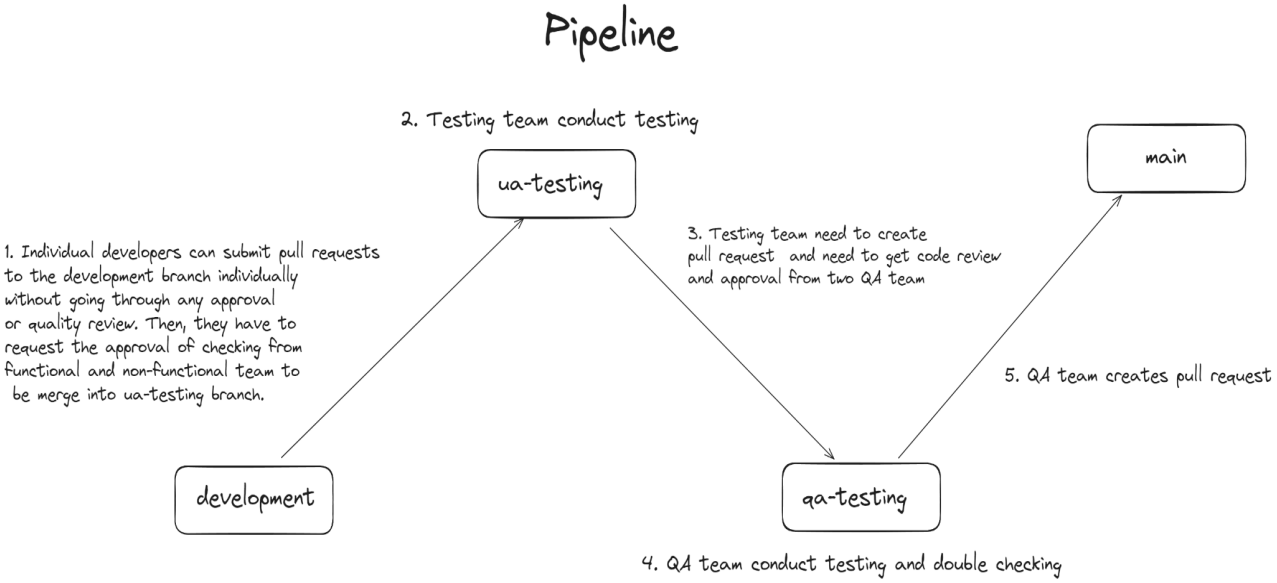
- When committing the changes, each file size must be less than 100 MiB.
- When adding a file to a repository via a browser, the file can be no larger than 25 MiB. (Illustrated as below).
- Prefer "forward-moving" git strategy (Avoid using git revert / git reset)



3.4 Pull request rules

No	Rule
1	Every pull request must acquire at least two (2) reviewers before merging.

3.5 Linear pipelines



Step	Description
1	Individual developers can commit their code to the development branch individually without going through any approval or quality review. Then, they have to assign the testing team member as assignee in pull request to act as a reminder for functional and non-functional team to merge the code into **ua-testing branch** and test the features out.
2	Functional and Non-functional Testing Team will conduct testing in this 'ua-testing' branch.
3	Testing team need to create pull request and assign quality assurance team member as assignee to get code review and approval from two Quality Assurance Team to merge into **'qa-testing'** branch.
4	The Quality Assurance Team conducts code review and double checking in 'qa-testing' branch.
5	The Quality Assurance Team creates pull requests to merge the code into the main branch.

4.0 References

About branches. (n.d.). GitHub Docs. Retrieved April 23, 2024, from https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/about-branches

About Repositories. (n.d.). GitHub Docs. Retrieved April 23, 2024 from https://docs.github.com/en/organizations/collaborating-with-groups-in-organizations/accessing-your-organizations-settings

About issues. (n.d.). GitHub Docs. Retrieved April 23, 2024 from https://docs.github.com/en/issues/tracking-your-work-with-issues/about-issues

About pull requests. (n.d.). GitHub Docs. Retrieved April 23, 2024 from https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/about-pull-requests

Accessing your organization's settings. (n.d.). GitHub Docs. Retrieved April 23, 2024, from https://docs.github.com/en/organizations/collaborating-with-groups-in-organizations/accessing-your-organizations-settings

Best practices for organizations. (n.d.). GitHub Docs. Retrieved April 23, 2024, from https://docs.github.com/en/organizations/collaborating-with-groups-in-organizations/best-practices-for-organizations

Creating a new repository. (n.d.). GitHub Docs. Retrieved April 23, 2024, from https://docs.github.com/en/repositories/creating-and-managing-repositories/creating-a-new-repository

Committing and reviewing changes to your project in GitHub Desktop (n.d.). GitHub Docs. Retrieved April 23, 2024, from

https://docs.github.com/en/desktop/making-changes-in-a-branch/committing-and-reviewing-changes-to-your-project-in-github-desktop

Roles in an organization. (n.d.). GitHub Docs. Retrieved April 23, 2024, from https://docs.github.com/en/organizations/managing-peoples-access-to-your-organization-with-roles/roles-in-an-organization