Title	GitHub Repository Automation and Health Management		
Type	Task Documentation	Date	07 May 2025

Purpose	This document systematically reports the tools and automation used by us to monitor and manage GitHub repository. It checks for stale branches, open pull requests, and security issues, while ensuring code quality and validation making it easier, faster, and more reliable.

GitHub Repository	Aparna2805/Project1

Task Team and Guide	Subramanian Lakshmi (Guide), Chirag Huria (PoC), Sajikumar Aparna, Priyadharshini Bharathi, Sahoo Divyaranjan

---

Task Overview

A. Build a reporting tool to give health status of a repository
Acceptance criteria
1.Identify stale branches more than 30 days with no commits
2. Identify open PRs for more than 7 days and list the reviewers with whom it is pending with
3. Email the report everyday

B. Build a git branch validator tool
Acceptance criteria for a valid feature branch:
1 Branch name should contain valid Jira ticket number
2. Branch name should follow pattern feature-<jira>-<short description>
3. Developers with commits to the branch should not have access to approve the PR

GitHub Repository Automation and Health Management

This system automates the identification of stale branches, open PRs, and reviewer tracking, while validating branch names with Jira ticket numbers and naming conventions, ensuring only eligible developers can approve pull requests.

Implementing the Deliverables -

A. **Health Status Reporting Tool**

Associated Files - staleBranches.py, prReviewers.py, mailReport.py (inside .github/identification)

Tasks Executed (In order to acceptance criteria)

A.1 Identifying Stale Repo – Developed a python script that uses the GitHub API to fetch branch details and commit timestamps, flags branches with no commits in the last 30 days, and includes them in the daily health report after validation.

A.2 Identify open PRs for more than 7 days and list the reviewers with whom it is pending: This script queries the GitHub API to retrieve open PRs, checks their creation date, and lists the pending reviewers for PRs older than 7 days for inclusion in the daily report.

A.3 Email the report every day: Scheduled a GitHub Action that triggers a Python script that compiles the stale branches and PR data, then sends an automated email with the repository's health status using an SMTP service.

Outcome Implemented stale branch identification, tracked open PRs, automated email reports, improved repository health through, and reduced manual effort in GitHub repository management.

B. Git branch validator tool

Associated Files – validate.py

Tasks Executed (In order to acceptance criteria)

B.1 Implemented Jira Ticket Validation: Ensured branch names include a valid Jira ticket number, preventing inconsistent naming using regex pattern.

B.2 Enforced Branch Naming Convention: Checked that branch names follow the feature-<jira-ticket>-<short-description> pattern to standardize branch creation.

B.3 Restricted PR Approval: Prevented developers who committed to a branch from approving their own pull requests, ensuring unbiased reviews.

Outcome Implemented Jira ticket validation in branch names, enforced standardized naming conventions, and prevented developers from approving their own pull requests.

C. Additional Tasks and Features Implemented Independently

Associated Files -

Tasks Executed (In compliance with acceptance criteria)

C.1 Adding a Branch : Added our own learning branch for branch creation and then created a development branch for ongoing work.

C.2 Conduct Code Quality Check: Implemented automated code quality checks using ######## to ensure code adheres to quality standards.

C.3 Implement GitHub Actions to Run Tests on Every Push/Pull Request: Configured GitHub Actions to automatically run unit and integration tests on every push or pull request, ensuring code stability.

C.4 Security Audit - Automate Alerts for Vulnerabilities Integrated security audit tools like ######## or automated scanning for vulnerabilities in dependencies, generating alerts when issues are detected.

C.5 Clean Branching Strategy & PR Checklist: Built a clean branching strategy with a checklist attached to each PR to ensure consistent naming, quality, and validation. PRs failing to meet the checklist requirements are automatically prevented from being merged.

C.6 Documenting Our Work: Properly documented the implemented processes, tools, command log and configurations for transparency and future reference.

Outcome Implemented clean branching strategy, automated code quality checks, GitHub Actions for testing, security vulnerability alerts, PR checklist enforcement, and documentation.

Repository Structure

```
+ hello.py                          # Initial Hello World file
|
+.github/
|
+ ── workflows/                     # GitHub Actions workflows
|    ├── ci.yml                     # CI/CD pipeline for test and validation (C3)
|    ├── healthReport.yml           # Daily health report email (A3)
|    ├── securityAudit.yml          # Security audit action (C4)
|    └── prChecklist.yml            # PR checklist enforcement (C5)
|
+ ── scripts/                       # Python Scripts
|    ├── staleBranches.py           # Script to identify stale branches (A1)
|    ├── prReviewers.py             # Script to fetch open PRs and list reviewers (A2)
|    └── mailReport.py              # Script to send daily report via email (A3)
|
+ ── validate-branch-name/          # Branch name validator tool
|    └── validate.py                # Script to validate branch name (B1, B2)
|
+ requirements.txt                 # Python dependencies
|
+.env                               # Environmental Variables (Github Credentials)
|
+.gitignore                         # To avoid leaking .env accidentally
|
+ README.md                         # Readme file
```

Conclusion -The implemented tools and processes enhance repository health and streamline workflows by automating code quality checks, tests, PR validation, and security audits for a more efficient, secure, and organized codebase.

Contributions	References

The terms A1 – C6 mentioned below refers to the above tasks executed. References are mentioned.

Name	**Document**	**References**
Aparna	A2,B3,C4,C5,C6	1. Quickstart for GitHub Actions - GitHub Docs 2. Workflow syntax for GitHub Actions - GitHub Docs 3. Best practices for naming Git branches 4. How to enforce code quality gates in GitHub Actions 5. How to set up branch protection rules in GitHub 6. How to Schedule Workflows in GitHub Actions | CICube
Bharathi	A3,B2,C2,C1,C6	
Divyaranjan	A1,B1,C3,C1,C6	

Documentation submission to	Lakshmi Subramanian