# Sterna Software Team GitHub Workflow Guidelines

**Version:** *v1.0.0*

**Prepared By:** *Guruprasath Balasubramanian, Software Team Lead*

## Introduction:

This document outlines the workflow, branching strategies, and GitHub practices for Sterna Security Devices Private Limited's software team. It is designed to be a comprehensive guide for team members who may be new to these concepts.

## Git Basics:

- **Repository:** A place where the code is stored.
- **Commit:** A snapshot of your code at a particular time.
- **Branch:** A parallel version of a repository.
- **Pull Request (PR):** A method of submitting contributions to the project.

## Git Commands:

- **Clone a repository:** *git clone REPOSITORY_URL*
- **Create a new branch:** *git checkout -b BRANCH_NAME*
- **Switch to a branch:** *git checkout BRANCH_NAME*
- **Commit changes:** *git commit -m "DESCRIPTION"*
- **Push changes:** *git push origin BRANCH_NAME*
- **Pull latest changes:** *git pull origin BRANCH_NAME*

## Branching Strategy:

### Main Branches:

- **main**: The production-ready branch. Only well-tested code should be merged into this branch.
- **dev**: The development branch, where all the features and bug fixes are merged before being tested and moved to the main branch.

### Other Branches:

- **feature:** For new features. These should be named as *feature/feature-name*.
- **bugfix:** For bug fixes. Name these as *bugfix/bug-description*.
- **hotfix:** For urgent fixes needed in the main branch. Name as *hotfix/hotfix-description*.

## **Working on a Feature:**

1. Checkout to the dev branch and pull the latest changes.
    - *git checkout dev*
    - *git pull origin dev*

2. Create a new branch for your feature: feature/feature-name.
    - *git checkout -b feature/feature-name*

3. Implement the feature. Add and commit your changes regularly.
    - *git add .*
    - *git commit -m "Implemented part of feature-name"*

4. Push the branch to the remote repository.
    - *git push origin feature/feature-name*

5. Create a PR to dev through the GitHub interface.

## **Working on a Bug Fix:**

1. Checkout to the dev branch and pull the latest changes.
    - *git checkout dev*
    - *git pull origin dev*

2. Create a new branch: bugfix/bug-description.
    - *git checkout -b bugfix/bug-description*

3. Fix the bug and commit your changes.
    - *git add .*
    - *git commit -m "Fixed bug-description"*

4. Push your changes.
    - git push origin bugfix/bug-description

5. Create a PR to dev.

## Working on a Hotfix:

1. Checkout to the main branch and pull the latest changes.
    - *git checkout main*
    - *git pull origin main*

2. Create a new branch: hotfix/hotfix-description.
    - *git checkout -b hotfix/hotfix-description*

3. Fix the issue urgently and commit your changes.
    - *git add .*
    - *git commit -m "Hotfix for hotfix-description"*

4. Push your changes.
    - *git push origin hotfix/hotfix-description*

5. Create a PR to main.

## Task Workflow:

1. **Create an Issue:** For every new task, create an issue in the repository. This issue serves as the primary reference for the task and contains all relevant information and discussions.

2. **Label the Issue:** Depending on the nature of the task, apply one of the following labels:
    - **Bug:** If the task is related to fixing an existing error or unexpected behavior.
    - **Feature:** If the task is related to implementing a new functionality or enhancing existing features.
    - **Hotfix:** If the task is an urgent fix that needs immediate attention.

3. **Branching Strategy:**
    - **Start from the dev Branch:** All tasks should be developed on a branch created from the dev branch. This ensures that work is based on the latest development version of the code.
    - **Name the Branch:** Branch names should be descriptive and follow the pattern of the task label, such as *feature/feature-name*, *bugfix/bug-description*, or *hotfix/hotfix-description*.

4. **Work on the Task:** Follow the development practices and guidelines detailed earlier in this document.

5. **Complete the Task:** Once the work is completed, reviewed, and tested, the changes should be merged into the appropriate branch (usually dev) following the team's code review process.

6. **Close the Issue:** After the task has been successfully merged, the corresponding issue can be closed, marking the task as completed.

## Issue Template:

***Issue Summary***
*[Provide a brief description of the issue]*

***Steps to Reproduce***
*1. [First step]*
*2. [Second step]*
*3. ...*

***Expected Behavior***
*[What you expect to happen]*

***Actual Behavior***
*[What actually happens]*

***Additional Comments***
*[Any additional information or context that could be helpful]*

**Pull Request Template:**

***Pull Request Summary***
*[Provide a brief description of the changes]*

***Related Issues***
*[Link to related issues if any]*

***Additional Comments***
*[Any additional information, comments, or context related to the PR]*

***Checklist***
*- [ ] Code compiles correctly*
*- [ ] I tested the functionality*
*- [ ] Added/updated unit tests*
*- [ ] Included meaningful comments*