Branching Strategy Documentation

1. Introduction

The purpose of this document is to outline the branching strategy used for project development. The strategy ensures efficient collaboration, clear version management, and a streamlined process for feature development, bug fixing, and release management.

2. Branch Types Overview

- **Main Branch (`main`)**: The primary branch representing the stable production version of the code. Only thoroughly tested and approved code should be merged into this branch.

- **Develop Branch (`develop`)**: The integration branch for features and fixes. This branch serves as the working branch for development and will include all completed features that are ready for testing.

- **Feature Branches (`feature/feature-name`)**: Branches created for specific features or tasks. They originate from the `develop` branch and must be merged back into `develop` after completion and code review.

- **Release Branches (`release/version`)**: Used to prepare the code for a production release. They are created from `develop` and allow for final testing, bug fixing, and release-specific changes.

- **Hotfix Branches (`hotfix/description`)**: Used for urgent bug fixes in the production code. They branch off `main` and are merged back into both `main` and `develop` after completion.

3. Branching Workflow

3.1. Creating a Branch:

Always create branches from the `develop` branch, except for hotfix branches, which originate from `main`. Naming conventions:
- Feature branches: `feature/feature-name`
- Release branches: `release/version`
- Hotfix branches: `hotfix/description`

3.2. Developing Features:

Create a new feature branch from `develop` using:
```bash
git checkout develop
git pull origin develop
git checkout -b feature/feature-name
```

Implement the feature with proper commit messages, following best practices for readability.
Push your branch to the remote repository and create a pull request (PR) when the feature is complete.

3.3. Code Review and Merging:

All feature branches must undergo a code review before merging.
PR approvals must be obtained from at least one other team member.
Merge the feature branch into `develop` using:

```bash
git checkout develop
git pull origin develop
git merge feature/feature-name
git push origin develop
```

3.4. Handling Merge Conflicts:

If conflicts arise, resolve them by:
- Using a code editor or IDE to review and edit conflicting files.
- Marking conflicts as resolved and committing the changes.

```bash
git add <resolved-files>
git commit
git push origin develop
```

3.5. Preparing for a Release:

Create a release branch from `develop`:

```bash
git checkout develop
git pull origin develop
git checkout -b release/version
```

Conduct final testing, bug fixes, and prepare the release notes.
Merge the release branch into both `main` and `develop` after the release.

3.6. Handling Hotfixes:

Create a hotfix branch from `main`:

```bash
git checkout main
git pull origin main
git checkout -b hotfix/description
```

Implement the fix, run tests, and push the changes.
Merge the hotfix branch into both `main` and `develop`:
```bash
git checkout main
git merge hotfix/description
git push origin main

git checkout develop
git merge hotfix/description
git push origin develop
```

4. Best Practices

- **Commit Messages**: Use clear and concise commit messages following the format: `type(scope): brief description`.
- **Rebasing**: Regularly rebase feature branches with `develop` to reduce merge conflicts.
- **Testing**: Ensure that automated and manual tests are conducted before merging into `develop` or `main`.
- **Documentation**: Document any significant changes or considerations relevant to the branch.

5. Conclusion

By following this branching strategy, the team can maintain a consistent, manageable workflow for development, release, and maintenance, ensuring high code quality and project stability.