Team GitHub Workflow Documentation

Introduction

This document outlines our team's GitHub workflow and branching strategy to ensure smooth collaboration among our 5 team members. By following these guidelines, we aim to maintain code quality, avoid conflicts, and create a seamless integration process.

Branching Strategy

We will use a modified Git Flow approach with the following branch types:

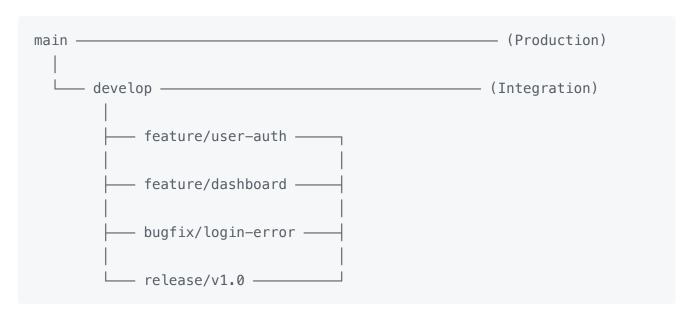
Core Branches

- main: Production-ready code. Only merged from develop after thorough testing.
- develop: Integration branch. Features are merged here after review.

Supporting Branches

- feature/[feature-name]: For new features or enhancements
- bugfix/[bug-name]: For bug fixes
- hotfix/[hotfix-name]: For critical production fixes
- release/[version]: For release preparation

Workflow Overview



Getting Started

Repository Setup

- 1. The team leader will create the main repository on GitHub.
- 2. All team members should:

- Clone the repository: git clone [repository-url]
- Set up their development environment

Initial Branches

The repository will be initialized with two branches:

- main: Contains production code
- develop: Where all feature branches will be merged

Daily Workflow

Starting Work on a New Feature

1. Make sure your local develop branch is up to date:

```
git checkout develop
git pull origin develop
```

2. Create a new feature branch:

```
git checkout -b feature/[feature-name]
```

3. Work on your feature, making regular commits.

Working on an Existing Feature

1. Switch to the feature branch:

```
git checkout feature/[feature-name]
```

2. Pull any updates if collaborating on the same feature:

```
git pull origin feature/[feature-name]
```

3. Continue working and committing changes.

Pushing Your Changes

1. Regularly push your feature branch to the remote repository:

```
git push origin feature/[feature-name]
```

2. If it's your first push for this branch:

```
git push -u origin feature/[feature-name]
```

Pull Request Process

When your feature is complete and ready for review:

1. Ensure your branch is up to date with develop:

```
git checkout develop
git pull origin develop
git checkout feature/[feature-name]
git merge develop
```

- 2. Resolve any conflicts if they arise.
- 3. Push your final changes:

```
git push origin feature/[feature-name]
```

- 4. Create a Pull Request (PR) on GitHub:
 - Go to the repository on GitHub
 - Click "Pull requests" > "New pull request"
 - Set the base branch to develop and compare branch to your feature branch
 - Fill out the PR template
 - Assign reviewers from the team
- 5. Address any feedback from reviewers by making additional commits to your feature branch.
- 6. Once approved, merge the PR:
 - Use the "Squash and merge" option to keep the commit history clean
 - Delete the feature branch after merging

Code Review Guidelines

For Authors

- Keep PRs focused and reasonably sized (ideally <300 lines of code)
- Provide context in the PR description
- Respond to comments promptly
- Be open to feedback

For Reviewers

- Review PRs within 24 hours when assigned
- Be constructive and respectful

- Focus on:
 - Code functionality
 - Code quality and best practices
 - Potential bugs or edge cases
 - Test coverage
- Use GitHub's suggestion feature for minor changes

Commit Message Guidelines

Follow the conventional commits specification:

```
<type>(<scope>): <short summary>
<body>
<footer>
```

Types:

feat : A new feature

• fix: A bug fix

docs: Documentation changes

style : Formatting changes

refactor: Code refactoring

test: Adding or fixing tests

chore: Maintenance tasks

Example:

```
feat(auth): implement user login functionality

Add login form with email and password validation.

Setup authentication service to handle login requests.

Closes #123
```

Merge Conflict Resolution

If you encounter merge conflicts:

1. First understand the conflicting changes:

```
git status
```

- 2. Resolve conflicts in each file:
 - Open the files with conflicts
 - Look for the <<<<<, '======', and >>>>> markers
 - Edit the files to resolve conflicts
 - Remove the conflict markers
- 3. After resolving all conflicts:

```
git add .
git commit -m "Merge develop into feature/[feature-name] and resolve
conflicts"
```

4. Consider pairing with a team member for complex conflicts.

Release Process

1. When the develop branch is ready for release:

```
git checkout develop
git pull origin develop
git checkout -b release/[version]
```

- 2. Make any final adjustments and version updates.
- 3. Create a PR to merge release/[version] into main.
- 4. After thorough testing and approval, merge to main.
- 5. Tag the release on main:

```
git checkout main
git pull origin main
git tag -a v[version] -m "Release v[version]"
git push origin v[version]
```

6. Also merge the release branch back to develop:

```
git checkout develop
git merge release/[version]
git push origin develop
```

7. Delete the release branch after merging.