**GitHub Standards**

Table of Contents

[**Commit Message Standards** 3](#_Toc203696232)

[**Format Structure** 3](#_Toc203696233)

[**Commit Types** 3](#_Toc203696234)

[**Commit Message Rules** 3](#_Toc203696235)

[**Commit Message Templates** 4](#_Toc203696236)

[**Pull Request Templates** 5](#_Toc203696237)

[**Setup Instructions** 5](#_Toc203696238)

[**Feature Template** 5](#_Toc203696239)

[**Bug Fix Template** 6](#_Toc203696240)

[**Hotfix Template** 6](#_Toc203696241)

[**Documentation Template** 7](#_Toc203696242)

[**Git Workflow Commands** 7](#_Toc203696243)

[**Basic Branch Operations** 7](#_Toc203696244)

[**Keeping Branches Updated** 8](#_Toc203696245)

[**Rebase Operations** 8](#_Toc203696246)

[**Merge Conflict Resolution** 9](#_Toc203696247)

[**Working with Remote Branches** 10](#_Toc203696248)

[**Stashing Changes** 11](#_Toc203696249)

[**Advanced Git Commands** 11](#_Toc203696250)

[**Branch Naming Conventions** 12](#_Toc203696251)

[**Format** 12](#_Toc203696252)

[**Examples** 12](#_Toc203696253)

[**Guidelines** 12](#_Toc203696254)

[**Code Review Guidelines** 12](#_Toc203696255)

[**Before Submitting PR** 12](#_Toc203696256)

[**Review Process** 13](#_Toc203696257)

[**Review Checklist** 13](#_Toc203696258)

**Commit Message Standards**

**Format Structure**

We follow the Conventional Commits specification, which provides a structured format for commit messages:

<type>[optional scope]: <description>

[optional body]

[optional footer(s)]

**Commit Types**

|  |  |  |
| --- | --- | --- |
| **Type** | **Description** | **Example** |
| feat | New feature for the user | feat: add user authentication |
| fix | Bug fix for the user | fix: resolve login redirect issue |
| docs | Documentation changes | docs: update API documentation |
| style | Code style changes (formatting, semicolons, etc.) | style: fix indentation in user.js |
| refactor | Code changes that neither fix bugs nor add features | refactor: extract payment logic |
| perf | Performance improvements | perf: optimize database queries |
| test | Adding or updating tests | test: add unit tests for auth module |
| build | Changes to build system or dependencies | build: update webpack config |
| ci | Changes to CI configuration | ci: add automated testing workflow |
| chore | Maintenance tasks | chore: update dependencies |
| revert | Reverting previous commits | revert: undo feature X implementation |

**Commit Message Rules**

Based on industry best practices, follow these guidelines:

1. **Subject Line (50 characters max)**
   * Use imperative mood ("Add" not "Added" or "Adding")
   * Capitalize the first letter
   * No period at the end
   * Be specific and descriptive
2. **Body (72 characters per line)**
   * Explain the "what" and "why", not the "how"
   * Separate from subject with a blank line
   * Use present tense
3. **Footer**
   * Reference issues: Closes #123
   * Breaking changes: BREAKING CHANGE: description

**Commit Message Templates**

**Basic Feature**

feat(auth): add OAuth2 integration

Implement Google OAuth2 authentication to allow users to sign in

with their Google accounts. This reduces friction in the signup

process and improves user experience.

- Add OAuth2 service configuration

- Create Google login button component

- Update user model to support OAuth users

Closes #456

**Bug Fix**

fix(api): handle null response in user endpoint

Prevent server crashes when user data is incomplete by adding

proper null checks and default values.

- Add null validation for user fields

- Return appropriate error messages

- Update tests to cover edge cases

Fixes #789

**Breaking Change**

feat(api)!: update user authentication flow

BREAKING CHANGE: The authentication endpoint now requires

a different request format. Update client applications to

use the new schema.

Migrates from basic auth to JWT tokens for improved security

and scalability.

Closes #234

**Documentation Update**

docs: add deployment guide

Create comprehensive deployment documentation including:

- Environment setup instructions

- Configuration examples

- Troubleshooting common issues

Closes #567

**Pull Request Templates**

**Setup Instructions**

Create these files in your repository:

**For single template:**

.github/PULL\_REQUEST\_TEMPLATE.md

**For multiple templates:**

.github/PULL\_REQUEST\_TEMPLATE/

├── feature.md

├── bugfix.md

├── hotfix.md

└── documentation.md

**Feature Template**

## 🚀 Feature

### Description

Brief description of the feature and its purpose.

### Changes Made

- [ ] Added new functionality

- [ ] Updated existing code

- [ ] Added/updated tests

- [ ] Updated documentation

### Type of Change

- [ ] New feature (non-breaking change which adds functionality)

- [ ] Breaking change (fix or feature that would cause existing functionality to not work as expected)

- [ ] This change requires a documentation update

### Testing

- [ ] Unit tests pass

- [ ] Integration tests pass

- [ ] Manual testing completed

- [ ] Cross-browser testing (if applicable)

### Screenshots/Demo

Include screenshots or GIFs demonstrating the new feature.

### Related Issues

Closes #[issue\_number]

### Additional Notes

Any additional information or context about the PR.

---

\*\*Reviewer Checklist:\*\*

- [ ] Code follows style guidelines

- [ ] Self-review completed

- [ ] Tests added/updated appropriately

- [ ] Documentation updated if needed

**Bug Fix Template**

## 🐛 Bug Fix

### Problem Description

Clear description of the bug that was fixed.

### Root Cause

Brief explanation of what caused the issue.

### Solution

Description of how the fix addresses the problem.

### Changes Made

- [ ] Fixed the identified issue

- [ ] Added regression tests

- [ ] Updated related documentation

### Testing

- [ ] Bug reproduction steps tested

- [ ] Unit tests pass

- [ ] Integration tests pass

- [ ] Manual verification completed

### Impact Assessment

- [ ] No breaking changes

- [ ] Backward compatible

- [ ] Safe to deploy

### Related Issues

Fixes #[issue\_number]

---

\*\*Reviewer Checklist:\*\*

- [ ] Fix addresses root cause

- [ ] No new bugs introduced

- [ ] Tests prevent regression

**Hotfix Template**

## 🚨 Hotfix

### Severity Level

- [ ] Critical (Production down)

- [ ] High (Major functionality affected)

- [ ] Medium (Minor functionality affected)

### Issue Description

Detailed description of the production issue.

### Fix Summary

Brief summary of the changes made.

### Verification

- [ ] Fix tested in staging environment

- [ ] Manual verification completed

- [ ] Monitoring alerts reviewed

### Deployment Plan

- [ ] Can be deployed immediately

- [ ] Requires coordinated deployment

- [ ] Needs database migration

### Rollback Plan

Description of how to rollback if issues arise.

### Post-Deployment Actions

- [ ] Monitor application logs

- [ ] Verify metrics return to normal

- [ ] Create follow-up tickets if needed

Fixes #[issue\_number]

---

\*\*Emergency Review Required:\*\*

@team-leads @on-call-engineer

**Documentation Template**

## 📚 Documentation

### Type of Documentation

- [ ] API documentation

- [ ] User guide

- [ ] Developer documentation

- [ ] README updates

- [ ] Code comments

### Changes Made

- [ ] Added new documentation

- [ ] Updated existing documentation

- [ ] Fixed documentation errors

- [ ] Improved clarity/structure

### Content Review

- [ ] Technical accuracy verified

- [ ] Grammar and spelling checked

- [ ] Links tested and working

- [ ] Examples tested

### Related Issues

Closes #[issue\_number]

**Git Workflow Commands**

**Basic Branch Operations**

**Create and Switch to New Branch**

# Create new branch from current branch

git checkout -b feature/user-authentication

# Create branch from specific commit

git checkout -b hotfix/login-bug abc123

# Create branch from main

git checkout main

git pull origin main

git checkout -b feature/new-feature

**Switch Between Branches**

# Switch to existing branch

git checkout main

git checkout feature/user-auth

# Switch using git switch (Git 2.23+)

git switch main

git switch feature/user-auth

# Create and switch in one command

git switch -c feature/new-feature

**Keeping Branches Updated**

**Update Main Branch**

# Fetch latest changes

git fetch origin

# Switch to main and update

git checkout main

git pull origin main

**Sync Feature Branch with Main**

# Option 1: Merge (preserves commit history)

git checkout feature/user-auth

git merge main

# Option 2: Rebase (cleaner history)

git checkout feature/user-auth

git rebase main

**Rebase Operations**

**Basic Rebase**

# Rebase current branch onto main

git rebase main

# Rebase specific branch

git rebase main feature/user-auth

# Rebase with specific commit

git rebase abc123

**Interactive Rebase (Clean Up Commits)**

# Rebase last 3 commits interactively

git rebase -i HEAD~3

# Rebase from specific commit

git rebase -i abc123

**Interactive Rebase Commands:**

* pick - Use the commit as is
* reword - Use commit but edit message
* edit - Use commit but stop for amending
* squash - Combine with previous commit
* drop - Remove the commit

**Rebase Feature Branch on Main**

# Complete workflow for rebasing feature branch

git checkout main

git pull origin main

git checkout feature/user-auth

git rebase main

# If conflicts occur, resolve them then:

git add .

git rebase --continue

# Force push rebased branch (only for feature branches)

git push --force-with-lease origin feature/user-auth

**Merge Conflict Resolution**

**Identify Conflicts**

# Check status during conflict

git status

# See conflicted files

git diff --name-only --diff-filter=U

**Resolve Conflicts Manually**

1. Open conflicted files in your editor
2. Look for conflict markers:
3. <<<<<<< HEADYour current branch changes=======Changes from other branch>>>>>>> branch-name
4. Edit to resolve conflicts
5. Remove conflict markers
6. Stage resolved files:
7. git add conflicted-file.js

**Complete Resolution**

# After resolving all conflicts

git add .

# For merge conflicts

git commit

# For rebase conflicts

git rebase --continue

# To abort merge/rebase

git merge --abort

git rebase --abort

**Advanced Conflict Resolution Tools**

# Use merge tool

git mergetool

# See differences during conflict

git diff

git diff --staged

# Show conflict in different format

git checkout --conflict=diff3 file.txt

**Working with Remote Branches**

**Push Changes**

# Push new branch to remote

git push -u origin feature/user-auth

# Push changes to existing remote branch

git push origin feature/user-auth

# Force push (use carefully!)

git push --force-with-lease origin feature/user-auth

**Pull Request Workflow**

# Before creating PR

git checkout main

git pull origin main

git checkout feature/user-auth

git rebase main # or merge main

git push origin feature/user-auth

# After PR is approved and merged

git checkout main

git pull origin main

git branch -d feature/user-auth # Delete local branch

git push origin --delete feature/user-auth # Delete remote branch

**Fetch and Pull**

# Fetch all remote changes without merging

git fetch origin

# Fetch specific branch

git fetch origin main

# Pull and merge

git pull origin main

# Pull and rebase

git pull --rebase origin main

**Stashing Changes**

**Basic Stash Operations**

# Stash current changes

git stash

# Stash with message

git stash push -m "Work in progress on login feature"

# List stashes

git stash list

# Apply latest stash

git stash pop

# Apply specific stash

git stash apply stash@{1}

# Drop stash

git stash drop stash@{0}

**Advanced Git Commands**

**Cherry-pick Commits**

# Cherry-pick specific commit

git cherry-pick abc123

# Cherry-pick range of commits

git cherry-pick abc123..def456

# Cherry-pick without committing

git cherry-pick --no-commit abc123

**Reset Operations**

# Soft reset (keep changes staged)

git reset --soft HEAD~1

# Mixed reset (keep changes unstaged)

git reset HEAD~1

# Hard reset (discard changes)

git reset --hard HEAD~1

# Reset to specific commit

git reset --hard abc123

**Useful Aliases**

Add these to your .gitconfig:

[alias]

co = checkout

br = branch

ci = commit

st = status

unstage = reset HEAD --

last = log -1 HEAD

visual = !gitk

lg = log --oneline --graph --decorate --all

please = push --force-with-lease

**Branch Naming Conventions**

**Format**

<type>/<description>

**Examples**

* feature/user-authentication
* bugfix/login-redirect-error
* hotfix/payment-gateway-down
* docs/update-api-documentation
* refactor/user-service-cleanup

**Guidelines**

* Use lowercase with hyphens
* Be descriptive but concise
* Include ticket number if applicable: feature/AUTH-123-oauth-integration
* Avoid special characters except hyphens and forward slashes

**Code Review Guidelines**

**Before Submitting PR**

* [ ] Self-review your code
* [ ] Ensure all tests pass
* [ ] Update documentation if needed
* [ ] Add appropriate commit messages
* [ ] Rebase on main branch
* [ ] Fill out PR template completely

**Review Process**

1. **Reviewer Assignment**: Assign 1-2 reviewers based on code area expertise
2. **Review Timeline**: Reviews should be completed within 24 hours
3. **Approval Requirements**: At least one approval required for merge
4. **CI/CD Checks**: All automated checks must pass

**Review Checklist**

* [ ] Code functionality works as intended
* [ ] Code follows team style guidelines
* [ ] No security vulnerabilities introduced
* [ ] Performance implications considered
* [ ] Tests adequately cover new/changed code
* [ ] Documentation updated appropriately
* [ ] No breaking changes (or properly documented)