Purpose of This Guide

This document provides a comprehensive guide to contributing to the Bloom Housing open-source project through GitHub. Our goal is to create a collaborative, efficient, and transparent development process that supports the project's mission of improving affordable housing access.

1. Repository Structure

Why Repository Structure Matters

The Bloom Housing project uses a monorepo-style repository with three main packages:

- api: Backend services and API
- sites: User-facing applications
- shared-helpers: Shared types and functions

Importance:

- Centralized codebase
- Easy dependency management
- Consistent code sharing across components
- Simplified project maintenance

2. Forking the Repository

Why Forking?

Forking creates a personal copy of the project that allows you to:

- Experiment without affecting the main project
- Propose changes safely
- Maintain a clean contribution history

Step-by-Step Process:

1. Navigate to the main Bloom Housing repository
2. Click "Fork" in the top right corner
3. Choose your personal GitHub account

Best Practices:

- Keep your fork updated with the main repository
- Use branches for specific features or fixes

3. Branching Strategy

Why a Structured Branching Approach?

A clear branching strategy helps:

- Isolate different pieces of work
- Maintain code quality
- Facilitate code reviews
- Reduce merge conflicts

Branch Naming Convention:

<type>/<descriptive-name>

Types:
- feature/: New functionality
- bugfix/: Fixing existing issues
- docs/: Documentation updates
- refactor/: Code restructuring
- test/: Adding or updating tests

Examples:

- feature/ai-application-guidance

- bugfix/mobile-responsive-layout
- docs/update-contribution-guidelines

4. Conventional Commits

Why Structured Commit Messages?

Conventional commits provide:

- Clear project history
- Automatic changelog generation
- Standardized communication
- Easier code review process

Commit Message Structure:

<type>(optional scope): <description>

[optional body]

[optional footer]

Commit Types:

- feat: New feature
- fix: Bug fix
- docs: Documentation changes
- style: Code formatting
- refactor: Code restructuring
- test: Adding/updating tests
- chore: Maintenance tasks

Example Commits:

feat(application): add AI-assisted application guidance

fix(listings): resolve mobile responsiveness issue for property cards

docs(readme): update contribution guidelines for new contributors

5. Pull Request (PR) Workflow

Why Pull Requests Matter

Pull requests are crucial for:

- Code review
- Maintaining code quality
- Facilitating team collaboration
- Ensuring project standards are met

Pull Request Checklist:

- [ ] Link associated GitHub Issue
- [ ] Describe changes thoroughly
- [ ] Include relevant screenshots
- [ ] Verify all tests pass
- [ ] Self-review code before requesting review

PR Description Template:

Description
[Provide a clear, concise description of changes]

Type of Change
- [ ] Bug fix
- [ ] New feature
- [ ] Breaking change

- [ ] Documentation update

How Tested
[Describe the tests performed]

Checklist
- [ ] Code follows project guidelines
- [ ] Added/updated tests
- [ ] Updated documentation
- [ ] Followed Responsible AI principles

6. Project Board and Issue Tracking

Why Use GitHub Projects?

GitHub Projects help:

- Visualize project progress
- Track work items
- Manage team workflow
- Improve project transparency

Recommended Board Columns:

- Backlog
- To Do
- In Progress
- Under Review
- Done

Issue Creation Guidelines:

- Use clear, descriptive titles
- Provide detailed context

- Add appropriate labels
- Link to project milestones

7. Responsible Contribution Principles

Why Ethical Considerations Matter

For Bloom Housing, we prioritize:

- Preventing bias
- Protecting user privacy
- Ensuring accessibility
- Maintaining transparency

Key Considerations:

- Use anonymized data
- Implement inclusive design
- Clearly explain AI decision-making
- Prioritize user control

8. Collaboration Best Practices

Effective Team Collaboration

- Communicate openly
- Review code constructively
- Share knowledge
- Support team learning
- Focus on project impact

Communication Channels:

- GitHub Discussions
- Team Slack/Discord

- Weekly check-ins

9. Merge Conflict Resolution

Why Conflict Prevention Matters

Preventing and resolving merge conflicts:

- Reduces development friction
- Maintains code quality
- Improves team productivity

Conflict Prevention Strategies:

- Communicate current work
- Frequently pull from main branch
- Break large features into smaller issues
- Use feature branches

Conflict Resolution:

```
# Fetch latest changes
git fetch upstream

# Rebase your feature branch
git checkout your-feature-branch
git rebase upstream/main

# Resolve conflicts manually
# Use your code editor or git tools

# Mark as resolved
git add .
git rebase --continue
```

```
# Push to your branch
git push -f origin your-feature-branch
```

10. Final Thoughts

Our Shared Mission

We're not just writing code—we're creating technology to make housing more accessible and equitable.

Key Principles:

- Be curious
- Stay humble
- Learn continuously
- Make a positive impact

Resources

- Bloom Housing GitHub Repository
- GitHub Docs
- Conventional Commits