# Product Team Training Manual

**Reviewed on: Aug 01 2025**

# Introduction:

This manual serves as a comprehensive guide to align the KAZIA product team with best practices, processes, and tools for efficient and high-quality product development. Our V1 app laid the groundwork, but challenges in scalability, security, and usability highlighted the need for a robust foundation. This document builds on those lessons, providing clear methodologies, collaborative workflows, and actionable guidelines to ensure consistent progress and impactful outcomes. It outlines the rules, frameworks, and steps needed to support the team in delivering a scalable, secure, and innovative platform aligned with KAZAPP vision and mission

*5Ps : **P**roper **P**lanning **P**revents **P**oor **P**erformance*

https://www.youtube.com/watch?v=hhuaVsOAMFc

# Onboarding Process

## Initial Tool Setup

- Access essential tools: GitHub, Trello, kumospace., Springboot, angular, postgresql,
- Learn project repositories and branch structure (GitFlow).

## Team Introductions

- Meet with the manager for project overview and team dynamics.
- Schedule one-on-one meetings with your new team mates to understand team roles and expertise.

## Documentation Review

- Review architecture diagrams, feature designs, and coding standards.
- Check design systems and reusable components for consistency.

# Agile Methodology

## Sprint Planning and Execution

- Plan sprints bi-weekly to define goals, prioritize tasks, and assign story points.
- Break tasks into subtasks with clear acceptance criteria.
- Track progress using Trello and ensure all team members have actionable items.

## Daily Standups

- Morning: Discuss completed work, plans for the day, and blockers.
- Evening: Summarize daily achievements and plan for the next day.

### Retrospectives

- Conduct every Monday to review: What went well, what didn't work, improvements for future sprints.

# Communication Standards

### Tools and Best Practices

- **Kumospace:** For team communication and real-time updates.
- **Trello:** To track tasks, log updates, and document decisions.
    - Best Practice: Write all task-related comments directly in Trello for Clarity and traceability.
- **GitHub Copilot/TabNine**: Al tools for faster coding and better suggestions.

### Stakeholder Engagement

- Hold regular meetings with stakeholders to stay aligned on goals and progress.
- Include stakeholder feedback in planning and development to meet their needs.
- Provide clear updates on features, timelines, and any challenges during meetings.
- Document and track stakeholder feedback in Jira for visibility and follow-up.

# Documentation and Code Management

### GitFlow and Version Control

- GitFlow: Use for managing branches and version control.
    - Main branch: For production-ready code.
    - Develop branch: For integrating features.
    - Feature branches: Use feature/ feature-name for new features.
    - Hotfix branches: Use hotfix/fix-name for urgent fixes.
- Always pull the latest changes from the develop branch before pushing your code.
- Keep staging and production branches updated to avoid conflicts.

### Commit Guidelines

- Include the Trello task ID in every commit message to track changes.
    - Example: git commit -m " Trello-1234: Fixed login issue "
- Write clear and simple commit messages that explain the changes.
- Combine related changes into one commit for Clarity.
- Don't commit commented-out code or unnecessary files.

### Best Practices

- Keep documentation up-to-date for:
    - Architecture diagrams and system designs.
    - Important technical decisions.
- Use GitHub or similar tools for hosting and sharing code.
- Document major updates in Trello tickets.
- Always review code in pull requests to ensure quality and consistency.

KAZIA solutions

# Feature Development Workflow

## Planning and Ticket Creation

- Discuss feature ideas with the team to understand requirements.
- Create detailed Trello tickets for each feature, including :
    - Clear goals and acceptance criteria.
    - User stories or problem descriptions.
    - Any supporting documents like designs or diagrams.
- Break large tasks into smaller, more manageable parts.

## Design and Refinement Phases

- Start with simple designs to align with stakeholders and engineers.
- Refine designs collaboratively with designers, engineers, and stakeholders.
- Finalize polished designs that are approved by the team.
- Attach finalized designs to the Trello tickets.

## Task Breakdown and Assignment

- Split features into smaller tasks, like backend work, frontend work, or QA testing.
- Assign tasks based on team members' skills and availability.
- Use story point estimation to size tasks:
    - Follow the Fibonacci sequence (1, 2, 3, 5, 8, 13) to estimate effort and complexity.
    - Discuss and assign story points as a team during sprint planning.
- Set clear priorities for each task.

## Best Practices

- Review and adjust story points if needed during sprints.
- Keep Jira tickets updated with progress and any changes to tasks or estimates.

# Quality Assurance (QA) Process

## Testing Protocols

- Test all completed tasks before marking them as Done.
- Ensure the feature meets:
    - The goals and criteria outlined in the ticket.
    - Design requirements and expected user behavior.
- Run unit tests, if they are set up, to validate individual components.
- Use AI-powered QA tools (e.g., Testim, Applitools) to automate test cases, identify issues, and streamline testing.
- Test in a staging environment to catch issues before release.
- Perform manual testing to find edge cases not covered by automated or AI tools.
- Log test results, including screenshots and test outcomes, directly in Trello tickets.

## Feedback and Iteration

- Provide clear and actionable feedback on any issues found:
    - Log bugs in Trello with detailed steps to reproduce.
    - Prioritize bugs based on their impact and urgency.
- Collaborate with developers to fix problems quickly.
- Retest fixed issues to ensure they are resolved.
- Regularly review and improve testing processes during retrospectives.

## Best Practices

- Run all available **unit tests** for each feature to catch issues early.
- Leverage AI tools to enhance testing speed and accuracy.
- Aim to have at least one **Pull Request** daily for code quality.
- Push tasks to the QA stage as soon as they are ready for consistent testing flow.
- Communicate blockers or critical issues immediately to avoid delays.
- Maintain detailed QA logs for reference and improvement.

# Pull Requests & Merging Guidelines

## Checklist for Pull Requests

1. **Code Quality**
    - Follow team coding standards.
    - Use clear names and write reusable, modular code.
    - Remove unnecessary or commented-out code.
2. **Testing**
    - Run all unit tests and check integration with existing features.
    - Highlight missing or incomplete tests.
3. **Documentation**
    - Update relevant documentation for new features or changes.
    - Add comments for complex logic.
    - Include architecture diagrams or design links if needed.
4. **Review Process**
    - All pull requests must be peer-reviewed.
    - Focus on code logic, performance, and maintainability.
    - Team leads approve after peer reviews are completed.
5. **Security**
    - Check that sensitive data is handled securely.
    - No hardcoded secrets, tokens, or keys.
    - Review API endpoints for vulnerabilities.

## Merging Process

1. Before Merging
   - Pass all tests and automated checks.
   - Ensure all review feedback is addressed.
   - Sync with the latest changes from the target branch.
2. How to Merge
   - Use squash merge to keep a clean commit history.
   - Ensure the code merges without conflicts.
   - Merge into develop for testing or main for production-ready code.
3. After Merging
   - Test the merged code in staging.
   - Close the pull request and update the related Jira ticket.
   - Notify the team or stakeholders of the changes.

## Best Practices

- Use tools like SonarQube for automated checks.
- Automate QA with tools like Testim.
- Prioritize performance, scalability, and security.
- Use feature toggles for gradual or experimental rollouts.