**Inventory and POS System - Process Documentation**

**Table of Contents**

1. [Introduction](#introduction)
2. [Development Process](#development-process)
   * [Agile Methodology](#agile-methodology)
   * [Sprint Planning](#sprint-planning)
   * [Daily Standups](#daily-standups)
   * [Sprint Review and Retrospective](#sprint-review-and-retrospective)
3. [Version Control Process](#version-control-process)
   * [Branching Strategy](#branching-strategy)
   * [Commit Message Guidelines](#commit-message-guidelines)
   * [Pull Request Workflow](#pull-request-workflow)
4. [Code Review Process](#code-review-process)
   * [Code Review Checklist](#code-review-checklist)
   * [Review Workflow](#review-workflow)
5. [Testing Process](#testing-process)
   * [Unit Testing](#unit-testing)
   * [Integration Testing](#integration-testing)
   * [End-to-End Testing](#end-to-end-testing)
   * [Continuous Integration](#continuous-integration)
6. [Deployment Process](#deployment-process)
   * [Staging Environment Deployment](#staging-environment-deployment)
   * [Production Environment Deployment](#production-environment-deployment)
   * [Rollback Procedures](#rollback-procedures)
7. [Maintenance and Support Process](#maintenance-and-support-process)
   * [Issue Tracking](#issue-tracking)
   * [Regular Updates](#regular-updates)
   * [User Support](#user-support)
8. [Appendix](#appendix)

**Introduction**

This document outlines the processes followed in the development, deployment, and maintenance of the Inventory and POS System. The purpose of this documentation is to ensure consistency, quality, and efficiency across all stages of the system’s lifecycle.

**Development Process**

**Agile Methodology**

The development of the Inventory and POS System follows the Agile methodology, which promotes iterative development, collaboration, and flexibility. Key principles include:

* **Customer Collaboration:** Regular feedback from stakeholders.
* **Adaptive Planning:** The ability to adjust plans based on project needs.
* **Early and Continuous Delivery:** Delivering functional components of the system early and often.

**Sprint Planning**

* **Duration:** Each sprint lasts two weeks.
* **Sprint Planning Meeting:** At the start of each sprint, the team meets to review the backlog, prioritize tasks, and define the sprint goal.
* **Task Estimation:** Tasks are estimated in story points to gauge effort and complexity.
* **Sprint Backlog:** The list of tasks committed to the sprint is documented in Jira.

**Daily Standups**

* **Duration:** 15 minutes.
* **Participants:** All development team members.
* **Focus:** Each member answers three questions:
  1. What did I work on yesterday?
  2. What will I work on today?
  3. Are there any blockers?

**Sprint Review and Retrospective**

* **Sprint Review:** At the end of each sprint, the team demonstrates the completed work to stakeholders.
* **Retrospective:** The team reflects on the sprint, discussing what went well, what didn’t, and how processes can be improved.

**Version Control Process**

**Branching Strategy**

* **Main Branches:**
  + main: Production-ready code.
  + develop: Latest developments ready for testing.
* **Feature Branches:**
  + Named as feature/feature-name.
* **Hotfix Branches:**
  + Named as hotfix/issue-name.

**Commit Message Guidelines**

* **Format:** <type>(<scope>): <subject>
* **Types:**
  + feat: A new feature.
  + fix: A bug fix.
  + docs: Documentation changes.
  + style: Code style changes (formatting, etc.).
  + refactor: Code refactoring without changing functionality.
  + test: Adding or updating tests.
  + chore: Maintenance tasks.
* **Example:** feat(inventory): Add stock alert notifications

**Pull Request Workflow**

* **Create Pull Request:** From feature branch to develop.
* **Reviewers Assigned:** At least two team members review the code.
* **Automated Checks:** All tests must pass before merging.
* **Approval:** Pull requests must be approved by reviewers before merging.

**Code Review Process**

**Code Review Checklist**

* **Code Quality:** Ensure the code adheres to coding standards and best practices.
* **Functionality:** Verify that the code meets the functional requirements.
* **Security:** Check for potential security vulnerabilities.
* **Performance:** Ensure the code is optimized for performance.
* **Testing:** Confirm that appropriate tests are included and passing.

**Review Workflow**

1. **Submit PR:** Developer submits a pull request.
2. **Assign Reviewers:** Two team members are assigned to review the PR.
3. **Review Comments:** Reviewers provide feedback or request changes.
4. **Address Feedback:** Developer addresses feedback and updates the PR.
5. **Final Approval:** Once all feedback is addressed, the PR is approved and merged.

**Testing Process**

**Unit Testing**

* **Framework:** Django’s built-in testing framework.
* **Location:** Tests are located in the tests/unit/ directory.
* **Goal:** Validate individual components (functions, models, views) in isolation.

**Integration Testing**

* **Location:** Tests are located in the tests/integration/ directory.
* **Goal:** Ensure that different modules work together as expected.
* **Tools:** Django Test Client for API testing.

**End-to-End Testing**

* **Tool:** Cypress.
* **Location:** Tests are located in the cypress/integration/ directory.
* **Goal:** Simulate real user scenarios to validate the system as a whole.

**Continuous Integration**

* **Tool:** GitHub Actions.
* **Process:** Every push to develop triggers automated tests. Only passing builds are eligible for deployment.

**Deployment Process**

**Staging Environment Deployment**

* **Frequency:** At the end of each sprint.
* **Process:**
  + Code is merged into the develop branch.
  + Automated tests are run.
  + If tests pass, the code is deployed to the staging environment.

**Production Environment Deployment**

* **Frequency:** Upon stakeholder approval of the staging environment.
* **Process:**
  + Code is merged into the main branch.
  + Final automated tests are run.
  + Deployment to production is triggered via GitHub Actions.

**Rollback Procedures**

* **Automated Rollback:** If a deployment fails, the system automatically rolls back to the previous stable version.
* **Manual Rollback:** In case of critical issues, the deployment can be manually rolled back using GitHub Actions or Docker.

**Maintenance and Support Process**

**Issue Tracking**

* **Tool:** Jira.
* **Process:**
  + Bugs and issues are reported by users or detected during testing.
  + Each issue is logged, prioritized, and assigned to a team member.
  + Progress is tracked until the issue is resolved and closed.

**Regular Updates**

* **Security Patches:** Applied as needed to address vulnerabilities.
* **Feature Enhancements:** Planned in future sprints based on user feedback and business needs.
* **System Maintenance:** Regular backups, performance monitoring, and optimization tasks.

**User Support**

* **Support Channels:** Users can reach support via email, phone, or the support portal.
* **Response Time:** Standard response time is within 24 hours for non-critical issues and immediate for critical issues.
* **Support Documentation:** User manuals and FAQs are available to assist with common issues.

**Appendix**

**Glossary**

* **CI/CD (Continuous Integration/Continuous Deployment):** A practice of automatically testing and deploying code changes.
* **PR (Pull Request):** A method of submitting contributions to a project, often used in Git-based workflows.
* **E2E Testing (End-to-End Testing):** Testing the flow of an application from start to finish to ensure all integrated components work as expected.

**References**

* **Django Documentation:** <https://docs.djangoproject.com/>
* **GitHub Actions Documentation:** <https://docs.github.com/en/actions>
* **Cypress Documentation:** https://docs.cypress.io/