**Inventory and POS System - Project Documentation**

**Table of Contents**

1. [Project Overview](#project-overview)
   * [Project Purpose](#project-purpose)
   * [Project Scope](#project-scope)
2. [Stakeholders](#stakeholders)
3. [Requirements](#requirements)
   * [Functional Requirements](#functional-requirements)
   * [Non-Functional Requirements](#non-functional-requirements)
4. [System Design](#system-design)
   * [Architecture Design](#architecture-design)
   * [Database Design](#database-design)
   * [User Interface Design](#user-interface-design)
5. [Development Plan](#development-plan)
   * [Timeline and Milestones](#timeline-and-milestones)
   * [Resources and Tools](#resources-and-tools)
   * [Risk Management](#risk-management)
6. [Testing and Quality Assurance](#testing-and-quality-assurance)
7. [Deployment Plan](#deployment-plan)
8. [Maintenance and Support](#maintenance-and-support)
9. [Appendix](#appendix)

**Project Overview**

**Project Purpose**

The Inventory and POS System is designed to streamline retail operations by providing a unified platform for managing inventory, processing sales, and generating reports. The system aims to increase efficiency, reduce errors, and provide real-time insights into business performance.

**Project Scope**

The scope of the project includes the development of a web-based system that:

* Manages product inventory, including stock levels and reorder points.
* Processes sales transactions through a user-friendly Point of Sale (POS) interface.
* Generates sales, inventory, and financial reports.
* Provides role-based access control to ensure security and data integrity.

**Stakeholders**

* **Project Sponsor:**
* **Project Manager:**
* **Development Team:**
  + Full Stack Developer: Emmanuel Otieno
* **End Users:**
  + Store Managers
  + Cashiers
  + Inventory Clerks
* **System Administrator:** Emmanuel Otieno

**Requirements**

**Functional Requirements**

1. **User Management:**
   * Users can log in using a username and password.
   * Roles include Admin and Cashier, with varying levels of access.
   * Admins can assign users roles in the system.
2. **Inventory Management:**
   * Ability to add, update, and delete products.
   * Track stock levels, set reorder points, and alert when stock is low.
   * Categorize products and manage product details like SKU, price, and description.
3. **Sales Management:**
   * Process sales transactions through the POS interface.
   * Support for multiple payment methods (cash, mobile payment).
   * Generate and print receipts for customers.
4. **Reporting:**
   * Generate sales reports by date range, product, or category.
   * Generate inventory reports showing stock levels, movements, and low-stock alerts.
   * Export reports in PDF.
5. **Payment Integration:**
   * Integration with payment gateways for processing mobile payments.
   * Secure handling of payment information, ensuring PCI compliance.

**Non-Functional Requirements**

1. **Performance:**
   * The system should handle up to 10,000 transactions per day without performance degradation.
   * Response time for critical operations (e.g., processing a sale) should not exceed 2 seconds.
2. **Security:**
   * All sensitive data, including passwords, must be encrypted.
   * Implement role-based access control (RBAC) to restrict access to system features.
   * Ensure compliance with relevant data protection regulations (e.g., GDPR).
3. **Usability:**
   * The system should have a user-friendly interface with intuitive navigation.
   * Provide training and documentation for end-users.
4. **Scalability:**
   * The system should be designed to scale horizontally to support additional stores or increased transaction volumes.
5. **Reliability:**
   * Ensure 99.9% uptime, with failover mechanisms in place for critical components.
   * Daily backups should be automated, with the ability to restore data within 30 minutes in case of failure.

**System Design**

**Architecture Design**

The system follows a three-tier architecture:

* **Presentation Layer:** Built with HTML, CSS, and JavaScript, using Django templates for dynamic content rendering.
* **Business Logic Layer:** Implemented in Python using the Django framework.
* **Data Layer:** MySQL is used as the primary database, handling all persistent storage for the system.

**Database Design**

The database consists of the following key tables:

* **Users:** Manages user accounts and roles.
* **Products:** Stores product details, including name, SKU, price, and stock levels.
* **Sales:** Records all sales transactions, including product IDs, quantities, and payment methods.
* **Inventory:** Tracks stock movements, including additions, removals, and current levels.

**User Interface Design**

The user interface (UI) is designed with usability in mind, featuring:

* **Dashboard:** Provides an overview of sales, inventory levels, and alerts.
* **Inventory Management:** A form-based interface for adding, editing, and deleting products.
* **POS Interface:** A simplified, touch-friendly interface for processing sales quickly.
* **Reporting:** A section dedicated to generating and viewing reports.

**Development Plan**

**Timeline and Milestones**

|  |  |  |
| --- | --- | --- |
| **Requirements Gathering** | **1 week** | Finalized requirements document |
| **System Design** | **2 weeks** | Completed system architecture and ERD |
| **Development** | **4 weeks** | Completed backend, frontend, and APIs |
| **Testing** | **1 week** | Completed unit, integration, and E2E tests |
| **Deployment** | **1 week** | System deployed to production |
| **User Training** | **1 week** | Training sessions completed |

**Resources and Tools**

* **Development Tools:** VS Code, Git
* **Database:** MySQL
* **Version Control:** GitHub
* **Project Management:** Jira
* **Communication:** Zoom
* **Testing Tools:** Selenium, Postman, Django Test Framework

**Risk Management**

|  |  |  |  |
| --- | --- | --- | --- |
| **Risk** | **Impact** | **Probability** | **Mitigation Strategy** |
| Delayed Requirements Gathering | High | Medium | Frequent check-ins with stakeholders |
| Data Breach | Critical | Low | Implement robust security measures, including encryption and regular security audits |
| Performance Issues | Medium | Medium | Conduct performance testing and optimize code where necessary |

**Testing and Quality Assurance**

* **Unit Testing:** Each module will be unit tested using Django’s built-in testing framework.
* **Integration Testing:** Integration tests will ensure that different modules work together seamlessly.
* **End-to-End Testing:** Automated E2E tests will simulate real user scenarios to ensure the system functions correctly from start to finish.
* **User Acceptance Testing (UAT):** End users will participate in UAT to validate that the system meets their needs.

**Deployment Plan**

* **Staging Environment:** The system will be deployed to a staging environment for final testing and approval.
* **Production Environment:** Upon successful testing, the system will be deployed to the production environment, with monitoring tools in place to ensure stability.

**Maintenance and Support**

* **Ongoing Support:** The development team will provide 6 months of post-launch support to address any issues or bugs.
* **System Updates:** Regular updates will be scheduled to address security vulnerabilities and add new features.
* **Documentation:** Comprehensive user and technical documentation will be provided to ensure smooth operation and maintenance of the system.

**Appendix**

**Glossary**

* **POS (Point of Sale):** A system where sales transactions occur.
* **SKU (Stock Keeping Unit):** A unique identifier for each product.
* **RBAC (Role-Based Access Control):** A system for managing user permissions based on their roles within the organization.

**References**

* **Django Documentation:** <https://docs.djangoproject.com/>
* **MySQL Documentation:** <https://dev.mysql.com/doc/>