# Inventory and POS System - Developer Documentation

**Table of Contents**

1. [**Introduction**](#introduction)
2. [**System Architecture**](#system-architecture)
   * [**Technology Stack**](#technology-stack)
   * [**High-Level Architecture**](#high-level-architecture)
   * [**Database Design**](#database-design)
3. [**Setup and Installation**](#setup-and-installation)
   * [**Prerequisites**](#prerequisites)
   * [**Local Development Environment**](#local-development-environment)
   * [**Production Environment**](#production-environment)
4. [**API Documentation**](#api-documentation)
   * [**Authentication**](#authentication)
   * [**Inventory Management Endpoints**](#inventory-management-endpoints)
   * [**POS Endpoints**](#pos-endpoints)
5. [**Frontend Development**](#frontend-development)
   * [**UI/UX Framework**](#uiux-framework)
   * [**Component Structure**](#component-structure)
   * [**State Management**](#state-management)
6. [**Backend Development**](#backend-development)
   * [**Business Logic**](#business-logic)
   * [**Error Handling**](#error-handling)
   * [**Logging**](#logging)
7. [**Testing**](#testing)
   * [**Unit Testing**](#unit-testing)
   * [**Integration Testing**](#integration-testing)
   * [**End-to-End Testing**](#end-to-end-testing)
8. [**Deployment and CI/CD**](#deployment-and-cicd)
   * [**Deployment Process**](#deployment-process)
   * [**CI/CD Pipeline**](#cicd-pipeline)
9. [**Security Considerations**](#security-considerations)
   * [**Data Encryption**](#data-encryption)
   * [**User Authentication and Authorization**](#user-authentication-and-authorization)
10. [**Version Control and Branching Strategy**](#version-control-and-branching-strategy)
11. [**Troubleshooting and Debugging**](#troubleshooting-and-debugging)
12. [**Contributing to the Project**](#contributing-to-the-project)
13. [**Appendix**](#appendix)
    * [**Entity-Relationship Diagram (ERD)**](#entity-relationship-diagram-erd)
    * [**API Endpoint Reference**](#api-endpoint-reference)

**Introduction**

**This document provides technical details for developers working on the Inventory and Point of Sale (POS) System. It covers the system architecture, setup instructions, API details, and guidelines for contributing to the project.**

**System Architecture**

**Technology Stack**

* **Frontend: HTML, CSS, JavaScript, Bootstrap 4**
* **Backend: Python (Django)**
* **Database: MySQL**
* **Authentication: JSON Web Tokens (JWT)**
* **API Documentation: Swagger/OpenAPI**
* **Deployment: Docker, Nginx, AWS EC2**

**High-Level Architecture**

**The system is designed with a modular architecture, separating the frontend, backend, and database layers.**

* **Frontend: The frontend communicates with the Django backend via RESTful APIs.**
* **Backend: The Django backend handles API requests, processes business logic, and interacts with the MySQL database.**
* **Database: MySQL is used for persistent storage of inventory and sales data. The database is normalized to reduce redundancy.**

**Database Design**

**The database consists of the following key tables:**

* **Users: Stores user information, including hashed passwords and roles.**
* **Products: Contains details of the products in the inventory.**
* **Sales: Records individual sales transactions.**
* **Inventory: Tracks stock levels and movements.**
* **Transactions: Logs payment information related to sales.**

**Sample Schema:**

**sql**

**CREATE TABLE `users\_user` (**

**`id` bigint(20) NOT NULL AUTO\_INCREMENT,**

**`password` varchar(128) NOT NULL,**

**`last\_login` datetime(6) DEFAULT NULL,**

**`is\_superuser` tinyint(1) NOT NULL,**

**`username` varchar(150) NOT NULL,**

**`first\_name` varchar(150) NOT NULL,**

**`last\_name` varchar(150) NOT NULL,**

**`is\_staff` tinyint(1) NOT NULL,**

**`is\_active` tinyint(1) NOT NULL,**

**`date\_joined` datetime(6) NOT NULL,**

**`email` varchar(254) DEFAULT NULL,**

**`role` varchar(50) NOT NULL,**

**`avatar` varchar(100) DEFAULT NULL,**

**`bio` longtext DEFAULT NULL,**

**`name` varchar(200) DEFAULT NULL,**

**PRIMARY KEY (`id`),**

**UNIQUE KEY `username` (`username`)**

**) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_general\_ci;**

**CREATE TABLE `products\_item` (**

**`id` bigint(20) NOT NULL AUTO\_INCREMENT,**

**`created\_at` datetime(6) NOT NULL,**

**`updated\_at` datetime(6) NOT NULL,**

**`name` varchar(255) NOT NULL,**

**`description` longtext DEFAULT NULL,**

**`price` decimal(10,2) NOT NULL,**

**`cost` decimal(10,2) NOT NULL,**

**`SKU` varchar(100) NOT NULL,**

**`barcode` varchar(100) DEFAULT NULL,**

**`category\_id` bigint(20) NOT NULL,**

**`image` varchar(100) DEFAULT NULL,**

**PRIMARY KEY (`id`),**

**UNIQUE KEY `SKU` (`SKU`)**

**) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_general\_ci;**

**CREATE TABLE `pos\_sale` (**

**`id` bigint(20) NOT NULL AUTO\_INCREMENT,**

**`created\_at` datetime(6) NOT NULL,**

**`updated\_at` datetime(6) NOT NULL,**

**`total\_amount` decimal(10,2) NOT NULL,**

**`customer\_contact` varchar(255) DEFAULT NULL,**

**`customer\_name` varchar(255) DEFAULT NULL,**

**`payment\_method` varchar(50) NOT NULL,**

**PRIMARY KEY (`id`)**

**) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_general\_ci;**

**CREATE TABLE `inventory\_inventory` (**

**`id` bigint(20) NOT NULL AUTO\_INCREMENT,**

**`created\_at` datetime(6) NOT NULL,**

**`updated\_at` datetime(6) NOT NULL,**

**`quantity` int(10) UNSIGNED NOT NULL CHECK (`quantity` >= 0),**

**`reorder\_level` int(10) UNSIGNED NOT NULL CHECK (`reorder\_level` >= 0),**

**`item\_id` bigint(20) NOT NULL,**

**`warehouse\_id` bigint(20) NOT NULL,**

**PRIMARY KEY (`id`)**

**) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_general\_ci;**

**Setup and Installation**

**Prerequisites**

* **Python: v3.8 or higher**
* **Django: v5.1 or higher**
* **MySQL: v8.x or higher**
* **Docker: Optional, for containerized deployment**

**Local Development Environment**

1. **Clone the Repository:**

**bash**

**Copy code**

**git clone https://github.com/emmanuelokoth2002/inventory\_management.git**

**cd inventory\_management.git**

1. **Create and Activate a Virtual Environment:**

**bash**

**Copy code**

**python3 -m venv venv**

**source venv/bin/activate**

1. **Install Dependencies:**

**bash**

**Copy code**

**pip install -r requirements.txt**

1. **Configure Environment Variables:**
   * **Create a .env file in the root directory and add the following:**

**makefile**

**Copy code**

**DB\_HOST=localhost**

**DB\_USER=root**

**DB\_PASS=password**

**DB\_NAME=inventory\_management**

**JWT\_SECRET=your\_jwt\_secret**

1. **Run Migrations:**

**bash**

**Copy code**

**python manage.py makemigrations**

**python manage.py migrate**

1. **Create a Superuser:**

**bash**

**Copy code**

**python manage.py createsuperuser**

1. **Start the Development Server:**

**bash**

**Copy code**

**python manage.py runserver**

**Production Environment**

* **Docker Deployment: Use the provided Dockerfile and docker-compose.yml to set up the production environment.**
* **Nginx Configuration: Configure Nginx as a reverse proxy to manage requests to the backend.**

**API Documentation**

**Authentication**

* **Endpoint: /api/auth/login**
* **Method: POST**
* **Description: Authenticates the user and returns a JWT token.**
* **Request Body:**

**json**

**Copy code**

**{**

**"username": "user123",**

**"password": "password123"**

**}**

* **Response:**

**json**

**Copy code**

**{**

**"token": "your\_jwt\_token\_here"**

**}**

**Inventory Management Endpoints**

* **Get All Products**
  + **Endpoint: /api/products/**
  + **Method: GET**
  + **Description: Retrieves a list of all products.**
  + **Response:**

**json**

**Copy code**

**[**

**{**

**"id": 1,**

**"name": "Product 1",**

**"description": "Ergonomic office chair",**

**"price": "10000.00",**

**"cost": "8000.00",**

**"SKU": "SKU123",**

**"barcode": "1234567890128",**

**"image": "http://localhost:8000/media/products/chair.jpg"**

**},**

**...**

**]**

* **Create a New Product**
  + **Endpoint: /api/products/add/**
  + **Method: POST**
  + **Description: Adds a new product to the inventory.**
  + **Request Body:**

**json**

**Copy code**

**{**

**"name": "New Product",**

**"description": "Description of the product",**

**"price": "200.00",**

**"cost": "150.00",**

**"SKU": "SKU456",**

**"barcode": "9876543210987",**

**"category\_id": 1,**

**"image": "http://localhost:8000/media/products/new\_product.jpg"**

**}**

* + **Response:**

**json**

**Copy code**

**{**

**"product\_id": 2,**

**"message": "Product created successfully"**

**}**

**Frontend Development**

**UI/UX Framework**

* **The system uses Bootstrap 4 for responsive design and styling. Custom styles are added in the assets/css/custom.css file.**

**Component Structure**

* **Django Templates:**
  + **base.html: Main template that includes the header, footer, and common elements.**
  + **dashboard.html: Displays the overview of sales, inventory levels, and alerts.**
  + **product\_list.html: Manages the display and CRUD operations for products.**
  + **pos.html: Handles the Point of Sale functionality.**
  + **inventory.html: Manages stock levels and reorder points.**

**State Management**

* **Django Context Processors: Used to pass common data to all templates.**
* **JavaScript: Custom JavaScript files manage dynamic elements and interactions within the frontend.**

**Backend Development**

**Business Logic**

* **The business logic is implemented in the services/ directory. Each service corresponds to a major component of the system (e.g., ProductService, SaleService).**

**Error Handling**

* **Centralized error handling is implemented using Django's middleware. Custom error classes are defined in utils/exceptions.py.**

**Logging**

* **Logging is handled using Django’s built-in logging framework. Logs are stored in the /logs/ directory and are rotated weekly.**

**Testing**

**Unit Testing**

* **Framework: Django’s built-in test framework**
* **Location: Unit tests are located in the tests/unit/ directory.**
* **Running Tests:**

**bash**

**Copy code**

**python manage.py test**

**Integration Testing**

* **Integration tests are located in the tests/integration/ directory.**
* **Use Django’s test client to test API endpoints.**

**End-to-End Testing**

* **Tool: Cypress (for frontend testing)**
* **Location: E2E tests are located in the cypress/integration/ directory.**
* **Running Tests:**

**bash**

**Copy code**

**npm run cypress:open**

**Deployment and CI/CD**

**Deployment Process**

* **Staging Environment: Push to the staging branch to trigger deployment to the staging server.**
* **Production Environment: Merge to the main branch to trigger deployment to the production server.**

**CI/CD Pipeline**

* **Tool: GitHub Actions**
* **Pipeline: The pipeline is defined in .github/workflows/deploy.yml.**
* **Stages:**
  + **Build: Installs dependencies and runs tests.**
  + **Deploy: Deploys the application using Docker to the target environment.**

**Security Considerations**

**Data Encryption**

* **All sensitive data, including passwords, are encrypted using Django’s PBKDF2PasswordHasher before storage.**
* **TLS/SSL is enforced for all network communication.**

**User Authentication and Authorization**

* **JWT tokens are used for user authentication.**
* **Role-based access control (RBAC) is implemented using Django’s built-in Groups and Permissions system to restrict access to certain features.**

**Version Control and Branching Strategy**

* **Main Branches:**
  + **main: Production-ready code.**
  + **staging: Latest tested features, ready for release.**
  + **development: Active development branch.**
* **Feature Branches:**
  + **Named following the pattern feature/feature-name.**
* **Pull Requests: All changes must be submitted via pull requests, which require code reviews before merging.**

**Troubleshooting and Debugging**

**Common Issues:**

* **Database Connection Errors: Ensure environment variables are correctly set and the MySQL server is running.**
* **API Errors: Review logs in the /logs/ directory for detailed error messages.**

**Debugging:**

* **Use Django’s DEBUG setting in settings.py to enable verbose logging and detailed error pages during development.**

**Contributing to the Project**

**Contribution Guidelines:**

* **Follow the guidelines outlined in CONTRIBUTING.md.**
* **Use consistent code formatting and adhere to PEP 8 standards.**
* **Write meaningful commit messages using the conventional commit format.**

**Appendix**

**Entity-Relationship Diagram (ERD)**

**API Endpoint Reference**

* **A complete list of API endpoints and their descriptions is available in the swagger.yaml file.**