**System Design Document(SDD)**

**Document Information**

| | |
|---|---|
| Project Name | PaintNova ColorLab System |
| Version | 1.0 |
| Date | 23/01/2026 |
| Authors | Abdalla Horera Fadhili<br>Sharifa Mohamed Ali<br>Amina Juma Hamad |

Contents

# 1. Introduction

## 1.1 Purpose

The System Design Document (SDD) explains the complete design of the PaintNova ColorLab Business Operations Management System by showing its architectural structure, system components, software modules, connection points, and data movement between components. The document transforms the approved Software Requirements Specification (SRS) into a technical blueprint that directs system development and system implementation and system maintenance activities.

## 1.2 Scope

The internal business operations system functions exclusively for use by PaintNova ColorLab staff members. The system enables users to process orders and perform operations for paint mixing and manage inventory and execute financial controls and create managerial reports. The system operates without direct customer interaction because customers must visit the physical store or call by phone to place their orders.

## 1.3 Design Goals

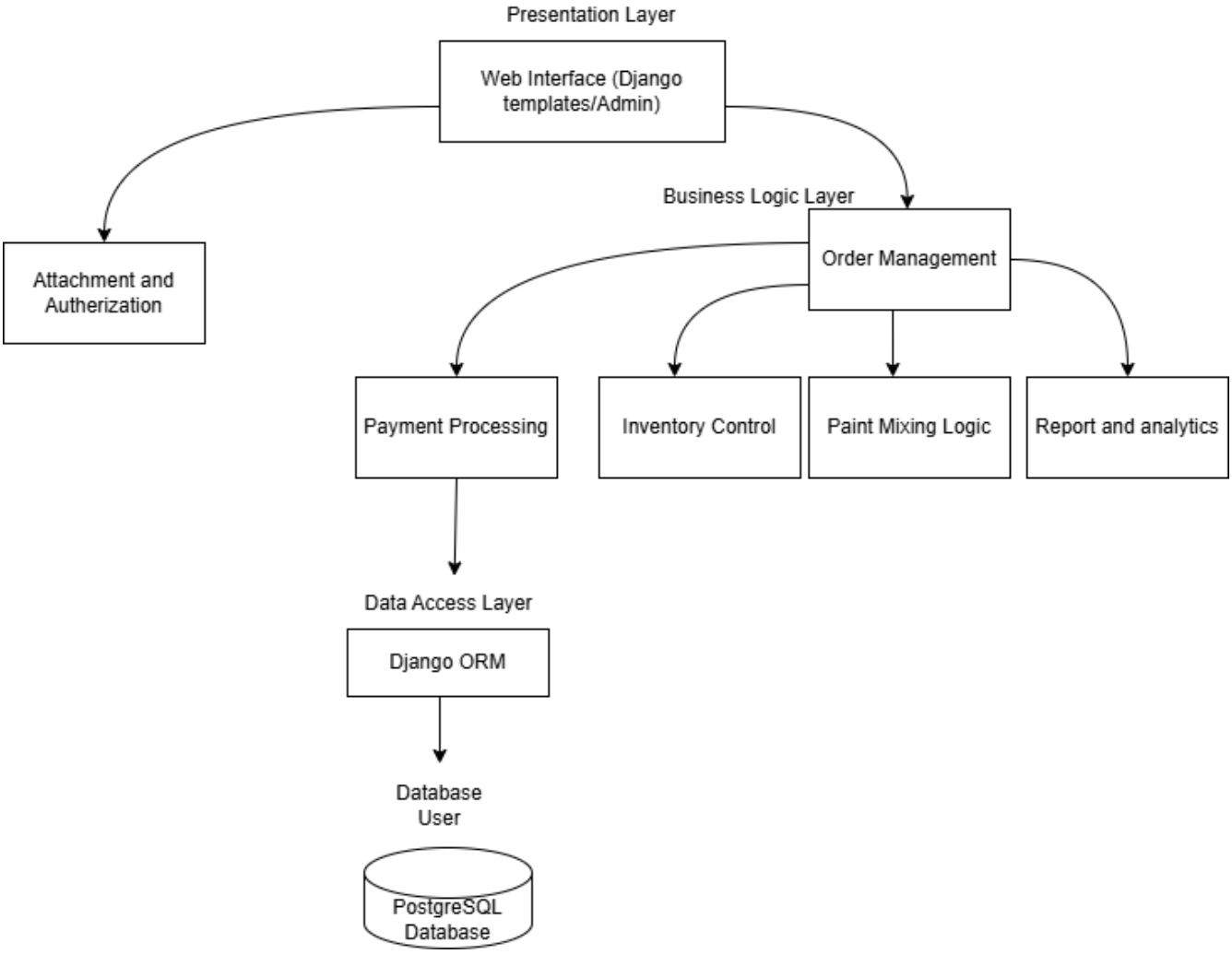
The system needs to process more incoming orders and handle greater amounts of inventory data. The system uses a modular and layered structure to achieve its maintenance requirements. The system needs to protect all business information and financial data which requires security. The system needs to deliver fast order processing and rapid inventory updates. The system requires an interface which allows non-technical staff to use it without difficulty.

# 2 System Architecture

## 2.1 Architecture Overview

Users access the system through a browser which operates according to web-based client–server architecture. The frontend system uses RESTful APIs to connect with the backend system while the backend system uses a relational database to store its data.

2.2 Architecture Diagrams




The organization selected the Layered (N-Tier) Architecture pattern as its architectural design pattern. The system contains four distinct layers which include:

1. Presentation Layer

2. Business Logic Layer

3. Data Access Layer

4. Database Layer The layered architecture enables better separation of concerns which simplifies system maintenance and provides better security while allowing upcoming system development to proceed without disrupting existing system components.

3. Technology Stack Selection

3.1 Frontend Technologies

- HTML5, CSS3, JavaScript
- Bootstrap enables the creation of websites that adapt to different screen sizes

3.2 Backend Technologiess

- Django serves as the Python framework for backend development
- The system uses the MVC architectural style which Django implements through its MTV Model Template View structure.

3.3 Database Selection

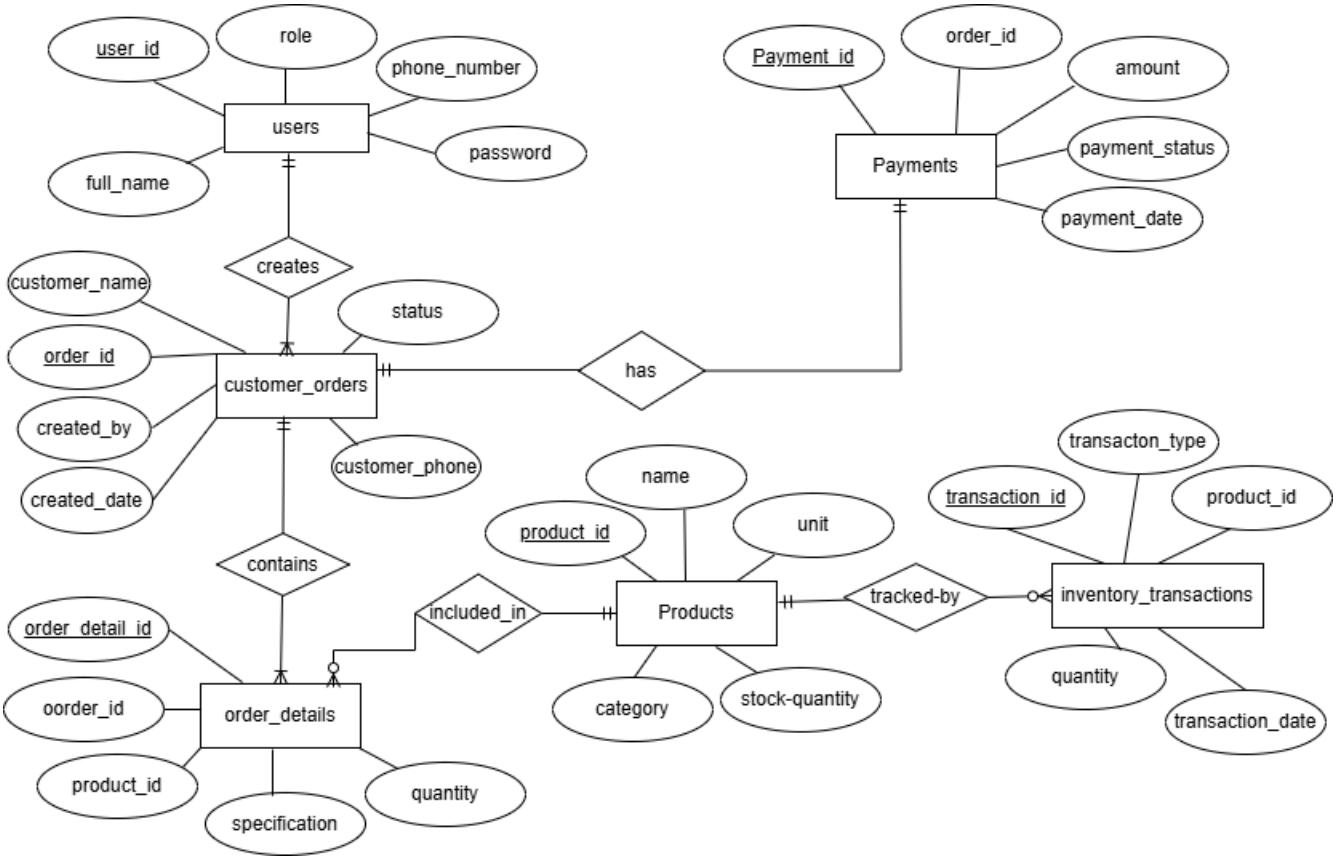
- PostgreSQL functions as a relational database system which serves as the database solution.

3.4 DevOps and Deployment

| **Tool** | **Purpose** |
|---|---|
| Git / GitHub | Version control |
| Docker | Containerization |
| Linux Server | Hosting environment |

4. Database Design

4.1 Entity Relational Diagram




4.2 Database Schema

**Table: Users**

- user_id (INT, PK)
- full_name (VARCHAR)
- role (VARCHAR)
- phone (VARCHAR)
- password_hash (VARCHAR)

**Table: Products**

- product_id (INT, PK)
- name (VARCHAR)

- category (VARCHAR)
- unit (VARCHAR)
- stock_quantity (INT)

**Table: Customer_orders**

- order_id (INT, PK)
- created_by (FK -> Users)
- status (VARCHAR)
- created_at (TIMESTAMP)

**Table: Order_Details**

- order_detail_id (INT, PK)
- order_id (FK ->Orders)
- product_id (FK -> Products)
- quantity (INT)
- specifications (TEXT)

**Table: Payments**

- payment_id (INT, PK)
- order_id (FK -> Orders)
- amount (DECIMAL)
- payment_status (VARCHAR)
- payment_date (DATE)

**Table: Inventory_Transactions**

- transaction_id (INT, PK)

- product_id (FK  -> Products)
- transaction_type (IN/OUT)
- quantity (INT)
- transaction_date (DATE)

4.3 Relationships

Users -> Orders (One to Many)

Customer_Orders -> Order_Details (One to Many)

Order_Details  -> Product(many to one)

Customer_Orders  -> Payments (One to One)

Products -> Inventory_Transactions (One to Many)

5. API Design

5.1 API Overview

The system exposes internal REST ful APIs for communication between the frontend and backend.

5.2 API Endpoints

Authentication Endpoints

| **Method** | **Endpoint** | **Description** | **Auth** |
|---|---|---|---|
| POST | /api/auth/login | User login | No |
| POST | /api/auth/logout | User logout | Yes |

Order Endpoints

| **Method** | **Endpoint** | **Description** | **Auth** |
|---|---|---|---|

| | | | |
|---|---|---|---|
| POST | /api/orders | Create order | Yes |
| GET | /api/orders | View orders | Yes |

5.3 Request/Response Examples

**POST /api/orders**

Request:

```
{
  "product_id": 3,
  "quantity": 10,
  "specifications": "Blue color mix"
}
```

Response:

```
{
  "success": true,
  "message": "Order created successfully"
}
```

6. User Interface Design

6.1 UI/UX Principles

• The design uses permanent elements together with identical color schemes.

• The system provides users with explicit confirmation through its feedback notifications.

• The system enables users to move through its features with straightforward pathways.

• The system uses a desktop-first approach for its initial design process.

6.2 Screen Layouts

1. Login Page - User authentication

2. Dashboard - Overview of orders and inventory

3. Order Page - Order entry and tracking

4. Inventory Page - Stock monitoring

6.3 Navigation Flow

The system navigation starts from login and goes to dashboard before accessing orders and inventory functions which lead to payment processing and finally ends at logout.

7. Security Design

7.1 Authentication

The system uses two methods for user access which include username and password login and secure session handling.

7.2 Authorization

The system uses Role-Based Access Control (RBAC) which includes three roles and their corresponding access rights.

7.3 Data Protection

The system protects data through three main security measures which include password hiding with bcrypt and secure web communication through HTTPS and monitoring user input through validation and Django ORM protection against SQL injection and XSS prevention.

8. Component Design

8.1 Component

1: Order Management

The system manages customer order processes by creating new orders and overseeing their entire lifecycle.

The system has three main duties which include recording new orders and checking product availability and system status tracking.

The system requires three components to function which include product component and inventory component and payment component.

The system uses RESTful APIs as its communication method with external systems.

9. Deployment Architecture

9.1 Deployment Diagram

The system operates through a centralized server system which connects its web server and application server and database server to create an integrated operational system.

9.2 Environment Configuration

The development environment operates through the Localhost system while the staging environment functions as an internal testing space and the production environment uses either on-premises or cloud-based servers for its operations.

10. Appendices

10.1 Glossary

The term API stands for Application Programming Interface while RBAC represents Role-Based Access Control and ORM means Object Relational Mapping.

10.2 Revision History

| **Version** | **Date** | **Author(s)** | **Changes** |
|---|---|---|---|
| 1.0 | 23/01/2026 | Amina Juma Hamad<br>Sharifa Mohamed Ali<br>Abdalla Horera Fadhili | Initial version |