

Code Structure Report

Generated by Stock Management Assistant

December 24, 2025

Contents

1	Introduction	2
2	File Analysis	2
2.1	models.py	2
2.2	manager.py	2
2.3	gui.py	3
2.4	interface.py	3
2.5	main.py	3

1 Introduction

This report outlines the code structure of the Stock Management project located in `e:/pythonProject/stock_management`. The project is organized into 5 main Python files covering the Data Model, Logic (Manager), Console Interface, GUI, and Entry Point.

2 File Analysis

2.1 models.py

This file defines the core data structures and enumerations used throughout the application to fetch and store entities.

- **Enums:**

- `OrderStatus`: Defines states like DRAFT, PENDING, CONFIRMED, CANCELLED, ARCHIVED.
- `PaymentStatus`: Defines states like UNPAID, PAID, PARTIALLY_PAID, REFUNDED.
- `DeliveryStatus`: Defines states like NOT_SHIPPED, SHIPPED, DELIVERED, RETURNED.
- `ProductStatus`: Defines states like ACTIVE, ARCHIVED.

- **Class Product:**

- **Attributes:** `code_prod`, `nom_prod`, `description`, `quantite`, `prix_unit`, `status`.
- **Methods:**
 - * `to_dict()`: Serializes product to dictionary.
 - * `from_dict(data)`: Deserializes dictionary to Product instance.
 - * `__str__()`: String representation.

- **Class OrderLine:**

- **Attributes:** `code_prod`, `quantity`, `price_at_order_time`.
- **Methods:**
 - * `total`: Property calculating line total price.
 - * `to_dict()`, `from_dict(data)`.

- **Class Order:**

- **Attributes:** `code_cmd`, `lines`, `status`, `payment_status`, `delivery_status`, timestamps (`created_at`, etc.), `paid_amount`.
- **Methods:**
 - * `total_amount`: Property calculating total order value.
 - * `to_dict()`, `from_dict(data)`.
 - * `__str__()`: String representation.

2.2 manager.py

This is the central controller (Logic Layer) handling business logic, data persistence (JSON and MySQL), and statistical calculations.

- **Class StockManager:**

- **Initialization:** `__init__` loads data from JSON files.
- **Data Persistence:**
 - * `load_data()`, `save_data()`
 - * `sync_data()`: Synchronizes data between local JSON and MySQL.
 - * `export_json_to_db()`: Exports local data to the database.
 - * `connect_db()`, `setup_database()`: Handles DB connection and table creation.

- **Product Management:**
 - * `add_product()`, `get_product()`, `update_product()`, `delete_product()`
 - * `get_all_products_sorted()`, `get_archived_products()`, `unarchive_product()`
- **Order Management:**
 - * `create_order()`, `add_line_to_order()`
 - * `confirm_order()`: Validates and changes status to CONFIRMED.
 - * `pay_order()`: Handles payments and stock deduction via `check_and_deduct_stock()`.
 - * `cancel_order()`, `delete_order()`, `unarchive_order()`
 - * `get_active_orders()`, `get_all_orders_history()`
- **Statistics & KPIs:**
 - * `get_dashboard_kpis()`: Returns total revenue, active orders, etc.
 - * `get_most_ordered_products()`, `get_revenue_over_time()`, `get_stock_levels()`, `get_order_status_distrib()`

2.3 gui.py

This file implements the Graphical User Interface using PyQt6.

- **Class MainWindow:**
 - The main container window implementing a customized dark theme.
 - **Methods:** `init_ui`, `refresh_app_data`, `on_tab_change`.
- **Class WelcomeTab:**
 - Displays the dashboard, KPIs, and synchronization controls.
 - **Methods:** `export_to_db`, `sync_data`, `import_from_db`.
- **Class ProductTab:**
 - Provides interface for CRUD operations on products.
 - **Methods:** `load_products`, `add_product`, `update_product`, `toggle_archive_all`, `toggle_archive_selected`.
- **Class OrderTab:**
 - Provides interface for creating and managing orders.
 - **Methods:** `load_orders`, `create_draft`, `add_line`, `confirm_order`, `pay_order`.

2.4 interface.py

Provides a Console/CLI user interface as an alternative to the GUI.

- **Class ConsoleInterface:**
 - **Menus:** `main_menu`, `product_menu`, `order_menu`.
 - **Views:** `list_products_view`, `stats_view`, `history_view`.
 - **Actions:** `add_product_view`, `update_product_view`, `delete_product_view`, `create_order_view`, `delete_order_view`.

2.5 main.py

The entry point of the application.

- Initializes `QApplication`.
- Creates and shows `MainWindow`.
- Starts the event loop.