Technical Description

C++ Store Management & Interaction System

Student 1: Eduard Ciulpan
Student 2: Alexandru Baba

May 29, 2025

1. Project Description

This project implements a **Store Management & Interaction System** split between two student developers:

- **Student 1 (Eduard Ciulpan)** is responsible for *managing the store*, which includes:
  - Adding, deleting, and modifying products (price, quantity) in the stock
  - Viewing all orders placed by customers
- **Student 2 (Alexandru Baba)** is responsible for *interacting with the store*, which includes:
  - Adding, modifying, and deleting products in/from a shopping cart
  - Creating an order by purchasing the contents of the shopping cart

The two parts communicate via text files that store products, orders, and the shopping cart. The applications do **not accept input from keyboard** and use only *command-line arguments* for interaction.

2. Data Structures

The following classes are used to model the data in the system:

Product Contains:

- `string barcode` — unique product identifier
- `string name` — product name
- `int quantity` — available quantity in stock or cart
- `double price` — unit price

Date Contains:

- `int day`
- `int month`

- `int year`

Order Contains:

- `vector<Product>` — list of products in the order
- `Date` — date when the order was placed

Class relationships: The `Order` class has a *composition* relationship with `Product`, because an order consists of multiple products. Additionally, `Order` aggregates a `Date` to indicate the order date.

3. File Structures

The two applications communicate and persist data through the following text files:

3.1. stoc.txt (Stock file)

Stores the current stock of products in the store.

```
<number_of_products>
<barcode_1> <name_1> <quantity_1> <price_1>
<barcode_2> <name_2> <quantity_2> <price_2>
...
```

3.2. comenzi.txt (Orders file)

Stores all placed orders.

```
<order_1_date: dd/mm/yyyy>
<order_1_product_barcode_1> <order_1_product_barcode_2> ...
<order_2_date: dd/mm/yyyy>
<order_2_product_barcode_1> <order_2_product_barcode_2> ...
...
```

3.3. cos$_{c}umparaturi.txt(Shoppingcartfile)$

Stores the products currently in the shopping cart.

```
<barcode_1> <quantity_1>
<barcode_2> <quantity_2>
...
```

4. Command-Line Commands

The two executables implement the following commands using `argv` parameters.

4.1. Application 1 (app_1.exe) — Store Management

- `view_stock_products`
  Displays all products currently in stock.
- `add_product <barcode> <name> <quantity> <price>`
  Adds a new product to the stock.
- `delete_product <barcode>`
  Deletes a product from the stock based on its barcode.
- `modify_product <price | quantity> <barcode> <new_value>`
  Modifies the price or quantity of an existing product.
- `view_orders`
  Displays all placed orders.

4.2. Application 2 (app_2.exe) — Shopping Cart Interaction

- `view_cart`
  Displays the current shopping cart contents.
- `add_product <barcode> <quantity>`
  Adds a product to the shopping cart.
- `modify_product <barcode> <new_quantity>`
  Changes the quantity of a product already in the cart.
- `delete_product <barcode>`
  Removes a product from the shopping cart.
- `purchase`
  Places an order based on the current shopping cart contents.

5. Additional Notes

- All input parameters are provided exclusively via **command-line arguments**. No runtime keyboard input (e.g., `std::cin`) is allowed.
- The class relationship between `Order` and `Product` fulfills the project requirement of having **at least two classes in a composition or aggregation relationship**.
- The system persists all data in text files allowing for communication and data sharing between the two executables.