|  |  |  |
| --- | --- | --- |
|  |  | Vending Machine Stock and User Management System  Project Report |

20.05.2025



## Prepared by:

Elif Sude Kürkçü

Emine Kalli

Eren Teper

Nesrin Kaya

Hakan Akavcı

**Abstract**

In this project, a vending machine application that includes user and administrator roles was developed using Java and JavaFX. While users can view and purchase products, administrators can perform product addition, stock and price update operations. The system is designed in a modular and extensible structure in accordance with object-oriented programming principles. Interaction with the database is provided in the background, and products are classified with filterable features. The project is a comprehensive software development study that integrates both technical application skills and real-world scenarios.

**Introduction**

The main purpose of this application is to add a digital layer to traditional vending systems, allowing users to select and purchase products remotely. The aim is to reduce the waiting time in front of the physical vending machine and to provide the user with a faster, more personalized shopping experience. Traditional vending machines are limited in terms of user experience: they have problems such as physically going to the vending machine, having only limited product information, and sometimes encountering payment problems. In addition, stock control and sales tracking may require manual operations for vending machine managers. This application was developed to eliminate these deficiencies on both the user and operator sides through digitalization.

What Problem Does It Solve?

**For Users:**

Being able to see and select products in advance

Contactless payment and fast delivery

Learning the location of the vending machine and viewing product stock status

**For Managers/Operators:**

Stock tracking and notification system

Analyzing sales data

Remote product update and pricing

Scope and Target User Audience

This application is designed specifically for vending machines in areas with heavy human traffic such as school campuses and universities.

**Target users:**

Individual users who want to shop from the vending machine

Operators operating the vending machine and who fill the stock

Managers who generally manage the system and who deal with authorization, reporting, etc.

**UML**

🔍 UML Class Diagram Description

Project Name: Vending Machine Stock and User Management System

Technology: JavaFX – PostgreSQL – OOP

Method: MVC-like Structure + Layered Architecture

**Database**

The PostgreSQL database was used in the application. The database structure was designed to meet the basic functions in the system. There are three main tables in the current database:

**operators**: It stores information such as operators' credentials, access authorizations and job descriptions.

**transaction\_history**: It stores the purchase transactions made by users, along with details such as date-time, product, payment information.

**products**: It contains information such as the names, prices, stock status and categories of the products in the vending machine.

The database is configured to run on a local server. However, if desired, the application can be updated to work with remote access by connecting the database to a cloud service (cloud platform). In this way, operators and managers can connect to the system from different devices and manage data simultaneously.

**User UML Explanation**

1. General Architecture and Structure

This UML Class Diagram shows the class structure and object relationships of a vending machine application developed with JavaFX. The system is divided into three main layers:

User Interface Layer (UI)

Business Logic Layer (Business Logic)

Data Access Layer (Data Access Layer)

The application basically consists of components that list products, update stocks, manage user inputs and communicate with the database.

🧱 2. Detailed Review of Classes

🟩 User

Role: It is the entry point of the JavaFX application. The main() method is defined here. The start (Stage) method starts the stage. It is responsible for starting other classes of the application, but it does not directly establish a relationship with other classes.

🟩 DatabaseManager

Role: It is a Singleton class that manages database connections and transactions.

Features:

url, user, password: Connection parameters

SQL queries are executed directly with methods such as getProductList(), saveTransaction(), checkCredentials().

executeBatchSql(): With this method, cost is optimized and all updated information is sent to the database at once.

Relationships:

Used by Inventory.

loadAllOperators() method loads Operator username and passwords. The operator is required for the manager to update the stock

🟩 Inventory

Role: Responsible for managing all products in the system, this class is not only a data holder, but also acts as a “hidden layer” between the user interface (UI) and the database management layer (DatabaseManager). Manages the entire product list and filtered sublists (drinks, foods, vegan products, etc.).

Features:

Automatic data synchronization with the GUI is provided using ObservableList, FilteredList.

Relationships:

Uses DatabaseManager.

InventoryItem objects are mapped to stock information.

🟩 InventoryItem

Role: Represents each product itself and its stock amount separately.

Relationships:

Inventory class manipulates this class (updates stock, pulls).

🟩 Product

Role: An abstract class that holds product information (name, price, stock). Subclasses (Food, Drink, Vegan, Sugarfree, NonEdible) are customized through inheritance.

🟩 ProductPieceController

Role: Manages product cells on the GUI.

🟩 UserSceneController

Role: Manages the user interface (for users who shop).

Relationships:

Uses Inventory.

Triggers operations such as adding products to the cart, purchasing, and operator's stock update in the GUI.

🟩 UpdateStockController

Role: Manages the operator interface that updates the stocks. Does not open the stock update screen without checking the operator's authorization. The operator must enter a username and password to access. Interacts with Inventory, DatabaseManager, Operator classes.

🟩 Operator

The class that holds the username and password information of the operators who log in to the system.

Relationships:

DatabaseManager class loads the operator list and performs authorization checks.

🟩 Food, Drink, Vegan, Sugarfree, NonEdible

Role: They are subclasses of the Product class, each of which defines its own type by overriding the displayInfo() method.

Note: Thanks to polymorphism, product types can be customized and displayed in the GUI.

✅3. Design Patterns and Software Principles

While control classes (UserSceneController, UpdateStockController) manage user interactions, the model (Product, InventoryItem) and the data layer (DatabaseManager) are separate. This class structure is not only an application framework; it is also an example of a sustainable, extensible and readable software architecture. All classes follow the Single responsibility principle. One class one responsibility. In addition, requests from the user interface go directly through Inventory. Instead of directly accessing the database, UI classes retrieve or update product information through the Inventory class. In this way, direct connections to the DatabaseManager are limited, and the dependency between layers in the system is reduced (low coupling). The DatabaseManager class is defined as a singleton object to prevent unnecessary resource consumption and data inconsistencies.

In the project, the Encapsulation principle has been effectively implemented. In all model classes (Product, Operator, InventoryItem etc.), fields are defined as private and external classes are prevented from directly accessing these fields. Instead, data access and update operations are performed via getter-setter methods.

Inheritance: Subclasses of the abstract Product class represent different types, providing type-based behavior in the GUI (Polymorphism). The equals method has been overridden in the Product class, the main purpose of which is to compare two objects based on content, not address. This method is a reflection of the polymorphism principle in Java because the equals method from the Object class has been redefined specifically for the Product class, and thus object behaviors have been made specific to the subclass.

Loose Coupling: Classes communicate with each other with minimum dependency, which increases the maintainability of the system.

This architecture created with JavaFX is planned as a modular, easy to maintain and future-proof system, while adhering to design principles.

**Admin UML Explanation**

1. Admin

The Admin class contains the start (Stage) method, which is the main entry point of the application. This class is derived from the JavaFX Application class and is responsible for starting the stage structure of the user interface.

2. LoginSceneController

This class controls the admin login scene. It asks for a password from the admin. The password received from the admin is compared with OperatorManager.adminPassword and verified.

3. AdminSceneController

This class manages the admin panel after login. The INTERFACE SIDE of operations such as viewing products in the panel, updating them, adding new products, and tracking account history, operators' passwords and permissions are carried out through this class.

4. AddProductFormController

This class controls the form scene where new products are processed.

5. SettingsController

This class allows the management of operator users in the system, and the updating of their usernames and passwords. At the same time, the transaction history is deleted from here. Before deletion, the administrator is asked for confirmation again.

6. DatabaseManager

All database operations are carried out centrally in this class. This class is suitable for singleton use. Thus, the database connection is established only once throughout the application. The connections throughout the entire application are designed to optimize (reduce / reduce) database expenditures to the maximum level.

7. OperatorManager

This is the class where all operators are managed. Operations such as adding a new operator, updating password and username are carried out through this class. allOperators: Statically stores all operators in the system.

loadAllOperators(): Loads all operators from the database.

8. Product

It is the product model of the application.

9. Operator

It is the model that represents the operator users in the system.

10. SafeIntegerConverter & SafeDoubleConverter

These classes are specialized in the TableView components used in AddProductFormController to convert the text (String) that the user enters in the table cell to numeric data types (Integer or Double).

StringConverter structures are required during TableCell editing in JavaFX.

The default built-in fromString() method is overridden by these classes, allowing only valid numerical expressions to be converted. In this way, when the user enters invalid input (for example "abc" or a space), NumberFormatException is not thrown, and system stability is maintained. With the return of null, invalid data is ignored by the system and the default behavior is maintained.

11. App utils

To maintain the readability of the code and to have a simple appearance, methods that are not directly related to the interface and classes are kept in the app utils class.

1-User Interface:

a-

metin, ekran görüntüsü, diyagram, sayı, numara içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

This is the main page of the user interface, where the order box, pay button, and current products are visible. Items that are out of stock appear grayed out.

b-

metin, ekran görüntüsü, sayı, numara, yazı tipi içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

The user can add products by clicking on them and remove them using the 'X' icon. The total price is displayed below the order box.

c-

ekran görüntüsü, metin, tasarım içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

After clicking the pay button, a feedback message is displayed.

d-

metin, ekran görüntüsü, diyagram, yazılım içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

Users can filter products by availability, dietary preferences, or specific ingredients. They can choose between food, drinks, and non-edible items. Vegan-friendly and sugar-free options are also available.

e-

metin, ekran görüntüsü, çizgi, yazı tipi içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

The operator can use the hidden button to update product stock.

f-

ekran görüntüsü, yazılım, tasarım içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

The operator is required to enter their username and password.

g-

metin, yazılım, sayı, numara, ekran görüntüsü içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

The hidden button enables the operator to update the stock, and the changes are saved afterwards.

2-Admin Interface:

a-

metin, ekran görüntüsü, yazılım, multimedya içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

Initially, the start page of the admin interface prompts the admin to enter a password. The admin can access the date, time, and details of orders; view all actions performed by users and operators; and access operator information such as names and passwords. In addition, the admin has the right to perform any operation on the database.

b-

metin, ekran görüntüsü içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

The admin can edit product names, categories, prices, and stock. On the right-hand side, buttons are provided to save changes, add a product, refresh the page, and download the transaction history.

c-  metin, ekran görüntüsü içeren bir resim

Yapay zeka tarafından oluşturulmuş içerik yanlış olabilir.

By clicking on a product, the admin can edit its details.

d-

ekran görüntüsü, metin içeren bir resim

Yapay zeka tarafından oluşturulmuş içerik yanlış olabilir.

The admin adds new products and sets their stock and price, by using the adding button.

e-ekran görüntüsü, metin içeren bir resim

Yapay zeka tarafından oluşturulmuş içerik yanlış olabilir.

The admin can view added products in the current product list.

f-

ekran görüntüsü, metin içeren bir resim

Yapay zeka tarafından oluşturulmuş içerik yanlış olabilir.

When the admin refreshes the page, a feedback message appears below.

g-yazılım, bilgisayar simgesi, multimedya yazılımı, metin içeren bir resim

Yapay zeka tarafından oluşturulmuş içerik yanlış olabilir. The transaction history can be exported as a CSV file.

h-metin, yazı tipi, ekran görüntüsü içeren bir resim

Yapay zeka tarafından oluşturulmuş içerik yanlış olabilir.

The admin can see the order history details here in CSV format.

j-

metin, multimedya yazılımı, yazılım, grafik yazılımı içeren bir resim

Yapay zeka tarafından oluşturulmuş içerik yanlış olabilir.

The including, adding and deleting operators and resetting passwords, are displayed above. Admin can also clear the transaction history.

k-

metin, ekran görüntüsü, sayı, numara, yazı tipi içeren bir resim

Yapay zeka tarafından oluşturulmuş içerik yanlış olabilir.

The confirmation message appears before the application exits.

**Conclusion**

The vending application developed within the scope of this project has provided significant convenience for both the user and the manager/operator by offering a digital solution to traditional vending systems. While users can view and purchase products through the application, it offers the advantage of contactless payment; managers and operators can easily access stock, sales and transaction history, which has increased the efficiency of the system. Thanks to the PostgreSQL database infrastructure, data is stored securely and integrated into cloud systems when needed, providing a more flexible use opportunity. Although the application has successfully achieved its basic goals, features such as adding comments to products, adding to favorites or campaign announcements can be integrated into the system in the future to further improve the user experience. This project has created a concrete example of how physical sales systems can be made more accessible and manageable in the digitalizing world.

**What Topics Did We Benefit From ChatGpt?**

1-We asked for suggestions for using Javafx.

2-We asked for the steps required to create styles.css files for the interface according to our own requests.

3-We wrote the values ​​received in the string data type to it in order to create the safeDoubleConverter and SafeIntegerConverter classes.

4-Chatgpt suggested that we should override the equals method in the Product class by debugging the code.

5-We wanted a variety to be provided for product filters and types.

6-We wanted a method to send the operations performed in the Database Manager class in bulk at once.

7-We wanted it to make the relationships that can be complex in UML clearer.