Supermarket Manager

Analysis and Design Document

Student:

**Group:**

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| <dd/mmm/yy> | <x.x> | <details> | <name> |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

I. Project Specification 4

II. Elaboration – Iteration 1.1 4

1. Domain Model 4

2. Architectural Design 4

2.1 Conceptual Architecture 4

2.2 Package Design 4

2.3 Component and Deployment Diagrams 4

III. Elaboration – Iteration 1.2 4

1. Design Model 4

1.1 Dynamic Behavior 4

1.2 Class Design 4

2. Data Model 4

3. Unit Testing 4

IV. Elaboration – Iteration 2 4

1. Architectural Design Refinement 4

2. Design Model Refinement 4

V. Construction and Transition 5

1. System Testing 5

2. Future improvements 5

VI. Bibliography 5

1. Project specification

The domain model for the Supermarket Manager application includes the following conceptual classes:

* User (abstract class)
  + Cashier
  + AdministratorUser
* Product
* Sale
* Invoice
* Report

The User class has a username and password, which are used for authentication. The Cashier class is responsible for selling products, while the AdministratorUser class is responsible for managing users, products, and generating reports.

The Product class has properties such as name, description, price, and stock level. The Sale class represents a single transaction and contains a list of products, the total price, and the time of the sale. The Invoice class represents the bill generated after a sale and contains information such as the customer's name, the products purchased, and the total price.

The Report class represents a report generated by the system and has properties such as the format (XML or TXT) and the content. Diagram

Description automatically generated

1. Architectural Design

2.1 Conceptual Architecture:

The Supermarket Manager application will use a three-tier client-server architecture, which separates the application into three layers:

* Presentation Layer: This layer is responsible for handling user interactions and displaying information to the user. It includes a web-based interface for both the regular and administrator users.
* Application Layer: This layer is responsible for implementing the business logic of the application. It includes the user and product management, sales processing, and report generation functionalities.
* Data Layer: This layer is responsible for managing the storage and retrieval of data. It includes a database management system like MySQL.

We will use the Model-View-Controller (MVC) architectural pattern, which separates the application into three interconnected components:

* Model: The model represents the application's data and business logic.
* View: The view is responsible for displaying the data to the user.
* Controller: The controller handles user input and manages the interaction between the model and the view.

Diagram, box and whisker chart

Description automatically generated

2.2 Package Design:

The Supermarket Manager application will be divided into the following packages:

* User Management: This package includes classes for managing user accounts, such as creating, reading, updating, and deleting user accounts.
* Product Management: This package includes classes for managing products, such as creating, reading, updating, and deleting products.
* Sales Management: This package includes classes for managing sales, such as adding products to the cart, calculating the total price, and generating invoices.
* Report Generation: This package includes classes for generating different types of reports, such as best-selling products, out-of-stock products, and employees' activity.
* Authentication and Authorization: This package includes classes for handling user authentication and authorization. Diagram

  Description automatically generated

2.3 Component and Deployment Diagrams:

The component diagram for the Supermarket Manager application includes the following components:

* Web Server: This component is responsible for handling HTTP requests from the client and returning responses. It includes the application server, which hosts the application logic.
* Database Server: This component is responsible for storing and retrieving data. It includes the MySQL database management system.
* Report Server: This component is responsible for generating reports and sending them to the client. It includes the report generation module.

The deployment diagram for the Supermarket Manager application includes the following nodes:

* Client: This node represents the user's device, which accesses the application through a web browser.
* Web Server: This node represents the server that hosts the web application and the application server.
* Database Server: This node represents the server that hosts the MySQL database management system.
* Report Server: This node represents the server that generates reports and sends them to the client.

# Elaboration – Iteration 1.2

# Design Model

## Dynamic Behavior

Sequence diagram:

A picture containing text, diagram, line, parallel

Description automatically generated

Communication diagram:

O imagine care conține text, de interior

Descriere generată automat

## Class Design

Singletone pattern for database connection

Observer pattern for notification propagation

Admins and users will have separate controllers that handle their operations, as well as separate views and layouts.

All the repositories Extend JpaRepository and use as a generic to link the database behind it all to the entities declared.

A screenshot of a computer screen

Description automatically generated with low confidence

A screenshot of a computer

Description automatically generated with medium confidence

# Data Model

A screenshot of a computer

Description automatically generated with low confidence

1 order can contain multiple products. Their total price will be calculated. Therefore, there is a one-to-many relation from order to product. One product has only 2 states: is part of an order(just 1) or not.

In table product, orderId is part of a foreign key and it also acts a as flag for revealing if the product is sold or not.

Product and type tables are in a many-to-many relationship, so an additional table was created to break down the dependency into 2 one to many relationships.

Table user has a field: “isAdmin” to distinguish regular users from admins.

# Elaboration – Iteration 2

Conceptual Architecture:

The conceptual architecture for the supermarket management application can be refined to include the following components:

1. User Interface (UI) Component:

* Responsible for providing an interface for users to interact with the application.
* Handles user input, displays information, and facilitates navigation between different screens.
* Provides separate views for regular users (cashiers) and administrator users.

1. Authentication Component:

* Responsible for managing user authentication and authorization.
* Verifies user credentials during login and grants appropriate access rights based on the user's role (cashier or administrator).

1. Sell Management Component:

* Handles the sell-related operations for regular users (cashiers).
* Includes functionality to initiate a sell, add products, compute the total price, decrease product stocks, finalize the sell, and generate invoices.

1. Inventory Management Component:

* Manages the supermarket's product inventory.
* Includes functionality to perform CRUD operations on products, track stock quantities, and update inventory levels.

1. User Management Component:

* Handles the management of user accounts for administrators.
* Provides CRUD operations to create, read, update, and delete user accounts.
* Ensures secure storage and retrieval of user account information.

1. Reporting Component:

* Responsible for generating reports based on different criteria, such as best-selling products, out-of-stock products, and employee activity.
* Supports generating reports in various formats (e.g., XML, TXT) for further analysis.

Package Design:

To refine the package design, we can follow package design principles such as cohesion and loose coupling. The components mentioned above can be organized into the following packages:

1. Presentation Package:

* Contains the User Interface (UI) Component.
* Provides screens, forms, and other UI elements for user interaction.

1. Authentication Package:

* Contains the Authentication Component.
* Handles user authentication and authorization logic.

1. Sell Package:

* Contains the Sell Management Component.
* Handles sell-related operations and maintains sell-related data.

1. Inventory Package:

* Contains the Inventory Management Component.
* Manages product inventory and stock information.

1. User Package:

* Contains the User Management Component.
* Handles user account management and stores user-related data securely.

1. Reporting Package:

* Contains the Reporting Component.
* Generates reports based on specified criteria and formats.

# Design Model Refinement

The refined UML class diagram for the application incorporates various class design principles and GRASP patterns to enhance its structure and maintainability. The diagram has been updated to ensure better encapsulation, cohesion, and responsibility assignment among the classes.

Presentation Layer

A screenshot of a computer program

Description automatically generated with low confidence

Service LayerA picture containing text, screenshot, diagram, font

Description automatically generated

Data Access Layer

A screenshot of a computer

Description automatically generated with medium confidence

To improve cohesion and adhere to the Single Responsibility Principle (SRP), each class has been assigned a specific responsibility and focuses on a distinct functionality within the application. Methods and attributes that do not directly relate to the class's responsibility have been moved to other classes responsible for handling those functionalities. This refinement enhances the maintainability of the classes and reduces code complexity.

The associations between classes in the diagram have been reviewed to accurately represent the relationships between objects in the system. Cardinalities and multiplicity have been adjusted accordingly to reflect the nature of these associations. Dependencies between classes have been identified, and where possible, dependency injection or interfaces have been utilized to decouple the classes. This decoupling improves flexibility, testability, and extensibility of the system.

# Construction and Transition

# System Testing

*[Describe how you applied integration testing and present the associated test case scenarios.]*

# Future improvements

*[Present future improvements for the system]*

Filters: Filter by sold product, filter by price, filter by date, multiple filters

Orders: Include products in orders

Artist Management: The application should allow artists to create accounts and upload their art pieces to the gallery, along with artist information such as biography, portfolio, and contact information.

Performance Optimization: Analyze the application's performance bottlenecks and optimize critical sections of the code or database queries to enhance overall responsiveness. Techniques such as query optimization, data indexing, and asynchronous processing can be employed to improve performance.

Automated Testing: Expand the test coverage by implementing automated unit tests, integration tests, and end-to-end tests. This ensures the stability and reliability of the application, reduces the occurrence of bugs, and facilitates easier maintenance and refactoring.

Notifications: Add multiple notifications, also on price change and product sales

# Construction and Transition

# System Testing

*[Describe how you applied integration testing and present the associated test case scenarios.]*

# Future improvements

*[Present future improvements for the system]*

# Bibliography

1. Diagrams:
2. <https://app.diagrams.net/>