Purchase Management System Documentation

Nikolozi Gagua

Lasha Japaridze

# 1. Description

The Purchase Management System is a desktop application built using Java, Swing, and MySQL. It is designed to help users manage customer purchases with comprehensive CRUD (Create, Read, Update, Delete) functionality across multiple entities—namely Customers, Products, Categories, and Invoices. The system provides an intuitive graphical user interface (GUI) that supports live stock management and search functionality. Key features include:  
- Managing customer details (insert, update, delete, search)  
- Managing product information with live stock updates  
- Organizing products using categories  
- Creating invoices by selecting customers and adding invoice items, with automatic calculation of totals and live stock adjustments  
- Integration via a Main Menu for seamless navigation between modules

# 2. Requirements

## Functional Requirements

- Customer Management: Add, update, delete, and search for customers.  
- Product Management: CRUD operations for products including live stock management.  
- Category Management: Manage product categories with CRUD operations.  
- Invoice Management: Create invoices and update stock levels automatically.

## Non-Functional Requirements

- Usability: User-friendly Swing-based GUI.  
- Reliability: Robust error handling and validations.  
- Maintainability: Modular DAO structure.  
- Performance: Responsive interface with proper transactions.

# 3. Database Tables (Structure & Data)

# A screenshot of a computer AI-generated content may be incorrect.

* A screenshot of a computer

  AI-generated content may be incorrect.Category table
* customer table

A close up of a contact us

AI-generated content may be incorrect.

* invoice table

A close up of numbers

AI-generated content may be incorrect.

* invoiceItem tabke

A screenshot of a computer

AI-generated content may be incorrect.

* product table

A screenshot of a phone number

AI-generated content may be incorrect.

# 4. ER Diagram

# A diagram of a product AI-generated content may be incorrect.

# 5. Interesting Source Code Snippets

**1.Email Validation in GUI**

📁 CustomerManagerGUI.java

if (!email.matches("^[\\w-.]+@([\\w-]+\\.)+[\\w-]{2,4}$")) {

showError("Please enter a valid email address");

return;

}

Ensures that users can’t submit malformed emails. Prevents dirty data in the database.

**2.Search Functionality with LIKE SQL Query**

📁 CustomerDAO.java

public static List<Customer> searchCustomers(String searchTerm) {

List<Customer> customers = new ArrayList<>();

String SQL = "SELECT \* FROM Customer WHERE name LIKE ? ORDER BY name";

try (Connection conn = DBConnection.connect();

PreparedStatement pstmt = conn.prepareStatement(SQL)) {

pstmt.setString(1, "%" + searchTerm + "%");

ResultSet rs = pstmt.executeQuery();

while (rs.next()) {

customers.add(new Customer(

rs.getInt("customer\_id"),

rs.getString("name"),

rs.getString("email"),

rs.getString("phone"),

rs.getTimestamp("created\_at")

));

}

} catch (SQLException e) {

System.err.println("Error searching customers: " + e.getMessage());

}

return customers;}

Enables fuzzy search (partial match) by using the % wildcard with SQL's LIKE. A small addition that makes the app much more user-friendly.

**3. Modular Database Connection With Encrypted Password**

📁 DBConnection.java

private static final String ENCRYPTED\_PASSWORD = "SjI0blZlQWNYIQ=="; // Base64-encoded

...

String password = new String(Base64.getDecoder().decode(ENCRYPTED\_PASSWORD));

connection = DriverManager.getConnection(URL, USER, password);

While it’s not perfect encryption, using Base64 encoding for the DB password adds a tiny layer of obscurity. Helps demonstrate some awareness of basic security practices.

**4. Dynamic Button Styling**

📁 CustomerManagerGUI.java

private JButton createStyledButton(String text, Color color) {

JButton button = new JButton(text);

button.setBackground(color);

button.setForeground(Color.BLACK);

button.setFocusPainted(false);

button.setFont(new Font("SansSerif", Font.BOLD, 12));

button.setPreferredSize(new Dimension(120, 30));

return button;

}

Enhances UX by making buttons look consistent and professional across the app. Reusable UI component approach.

# 

# 6. Tests

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **Test Action** | **Expected Result** | **Pass/Fail** |
| Customer Insertion | Insert a customer with valid data | New customer appears in the table and database | Pass |
| Customer Email Validation | Try to insert invalid email format | Error message is shown | Pass |
| Customer Phone Validation | Enter phone number with invalid characters | Error message is shown | Pass |
| Customer Update | Update an existing customer's email | Email updated in table and database | Pass |
| Customer Deletion | Delete a selected customer | Customer removed from table and database | Pass |
| Product Insertion | Add a new product with stock | Product shows up in product list | Pass |
| Live Stock Update | Create an invoice using a product | Product stock reduces by purchased quantity | Pass |
| Invoice Creation | Add multiple products to an invoice | Invoice saved with correct total and items | Pass |
| Search Product | Search for a product by name | Filtered product list is displayed | Pass |
| Search Customer | Search for a customer by name | Filtered customer list is displayed | Pass |
| Category Validation | Use invalid category ID | System shows category not found or error | Pass |