BONAFIDE CERTIFICATE

This is to certify that this project report titled
'Textiles Store Management
System' is a project work of
R. Vinoothna
bearing roll no. 1602-22-737-128
who carried out.
this project under my supervision in
the IV semester
of the academic year 2023- 2024

Signature
External Examiner

Signature
Internal Examiner

CONTENTS:

Problem Statement:

A textile store requires an efficient management system to handle inventory, sales, and customer information. The current manual system is prone to errors and lacks the capability to provide real-time insights into stock levels and sales data. To address these issues, a software solution is needed that automates inventory management, facilitates bill generation, and enables easy access to customer and supplier information.

Abstract:

The Textiles Store Management System is a software solution designed to streamline the operations of a textile store. It provides functionalities for managing inventory, generating bills, and maintaining customer and supplier records. The system aims to eliminate manual errors, improve efficiency, and enhance decision-making by providing real-time insights into stock levels and sales data.

Design Requirements:

List of tables, its attributes and their domains:

1)Table-1: Stock Table

- stock_id (Primary Key)
- stock_name
- quantity
- price

2)Table-2: Customer Table

- customer_id (Primary Key)
- customer_name
- contact_info
- address

3)Table-3: Purchase Table

- purchase_id (Primary Key)
- customer_id (Foreign Key referencing Customer)
- stock_id (Foreign Key referencing Stock)
- quantity
- total_amount

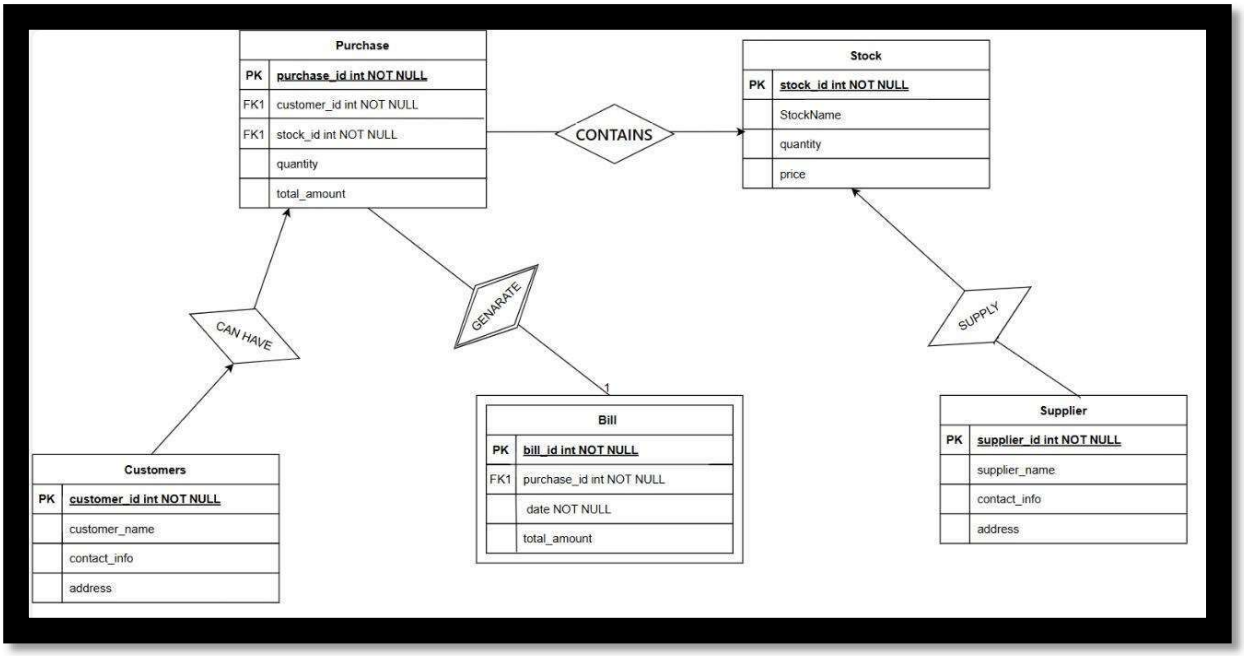
4)Table-4: Supplier Table

- supplier_id (Primary Key)
- supplier_name
- contact_info
- address

5)Table-5: Bill Table

- bill_id (Primary Key)
- purchase_id (Foreign Key referencing Purchase)
- total_amount

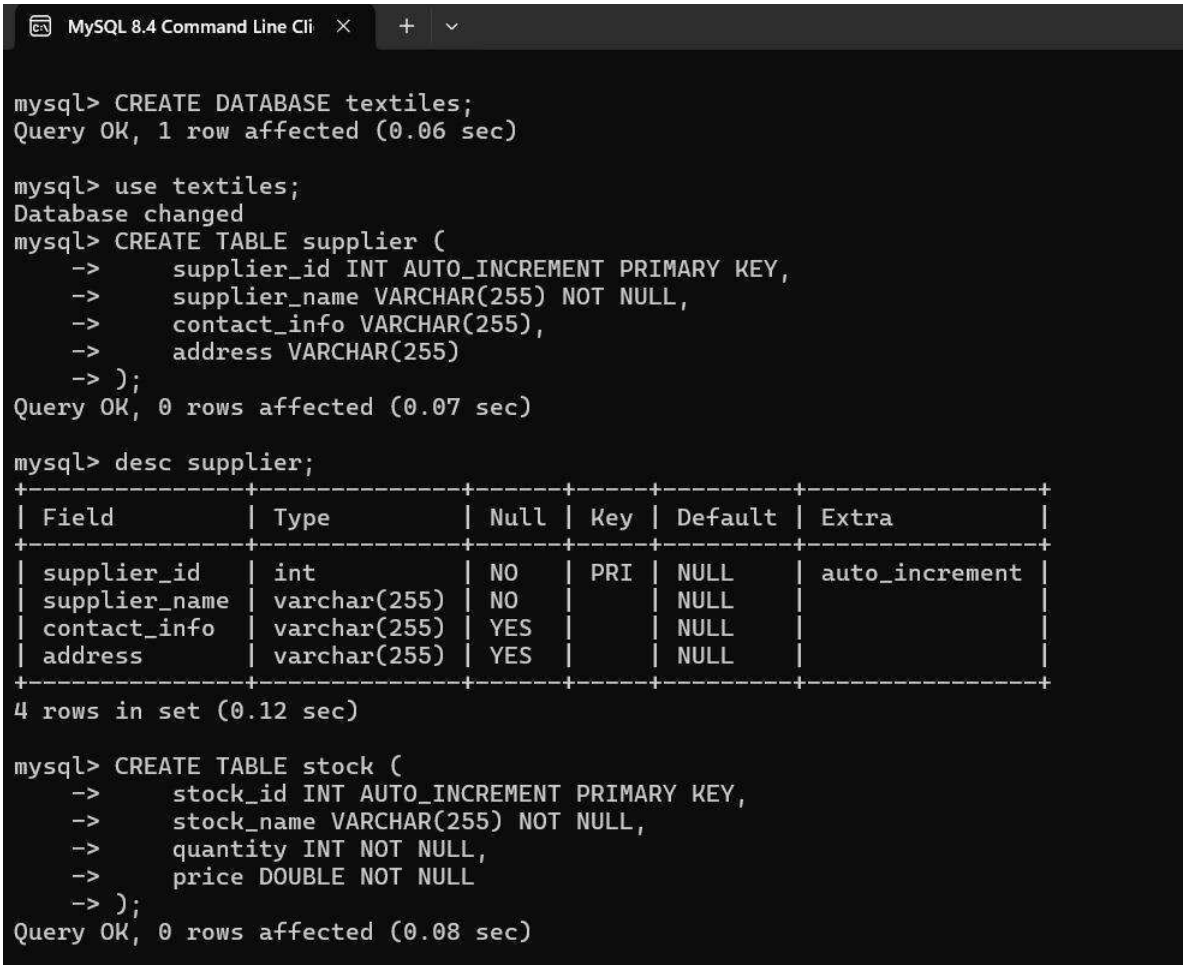
ER DIAGRAM:


Purchase
PK purchase_id int NOT NULL
FK1 customer_id int NOT NULL
FK1 stock_id int NOT NULL
quantity
total_amount
CONTAINS
Stock
PK stock_id int NOT NULL
StockName
quantity
price
CAN HAVE
GENARATE
SUPPLY
1
Customers
PK customer_id int NOT NULL
customer_name
contact_info
address
Bill
PK bill_id int NOT NULL
FK1 purchase_id int NOT NULL
date NOT NULL
total_amount
Supplier
PK supplier_id int NOT NULL
supplier_name
contact_info
address

MAPPING CARDINALITIES:

1. **Customer and Purchase**
    - **Cardinality**: One-to-Many (1:N)
    - **Explanation**: One customer can make multiple purchases, but each purchase is made by one customer.
2. **Product and Purchase**
    - **Cardinality**: Many-to-Many (M:N)
    - **Explanation**: One product can be part of multiple purchases, and one purchase can include multiple products. This relationship usually requires a junction table (e.g., PurchaseDetails) to manage the many-to-many relationship.
3. **Purchase and Bill**
    - **Cardinality**: One-to-One (1:1)
    - **Explanation**: Each purchase generates one bill, and each bill corresponds to exactly one purchase. This implies a dependent or weak relationship where the Bill entity relies on the Purchase entity for its existence.
4. **Supplier and Product**
    - **Cardinality**: One-to-Many (1:N)
    - **Explanation**: One supplier can provide multiple products, but each product is supplied by one supplier.
5. **Employee and Purchase**
    - **Cardinality**: One-to-Many (1:N)
    - **Explanation**: One employee can handle multiple purchases, but each purchase is managed by one employee.
6. **Product and Inventory**
    - **Cardinality**: One-to-One (1:1)
    - **Explanation**: Each product has a unique inventory record, and each inventory record corresponds to one product. This might be considered more of an attribute of the product rather than a separate entity in some designs.

DDL COMMANDS :




MySQL 8.4 Command Line Cli
mysql> desc stock;
+------------+--------------+------+-----+---------+----------------+
| Field | Type | Null | Key | Default | Extra |
+------------+--------------+------+-----+---------+----------------+
stock_id	int	NO	PRI	NULL	auto_increment
stock_name	varchar(255)	NO		NULL	
quantity	int	NO		NULL	
price	double	NO		NULL	
+------------+--------------+------+-----+---------+----------------+
4 rows in set (0.07 sec)
mysql> CREATE TABLE customer (
-> customer_id INT AUTO_INCREMENT PRIMARY KEY,
-> customer_name VARCHAR(255) NOT NULL,
-> contact_info VARCHAR(255),
-> address VARCHAR(255)
->);
Query OK, 0 rows affected (0.11 sec)
mysql> desc customer;
+---------------+--------------+------+-----+---------+----------------+
| Field | Type | Null | Key | Default | Extra |
+---------------+--------------+------+-----+---------+----------------+
customer_id	int	NO	PRI	NULL	auto_increment
customer_name	varchar(255)	NO		NULL	
contact_info	varchar(255)	YES		NULL	
address	varchar(255)	YES		NULL	
+---------------+--------------+------+-----+---------+----------------+
4 rows in set (0.00 sec)
mysql> CREATE TABLE purchase (
-> purchase_id INT AUTO_INCREMENT PRIMARY KEY,
-> customer_id INT,
-> stock_id INT,
-> quantity INT NOT NULL,
-> total_amount DOUBLE NOT NULL,
-> FOREIGN KEY (customer_id) REFERENCES customer(customer_id),
-> FOREIGN KEY (stock_id) REFERENCES stock(stock_id)
->);
Query OK, 0 rows affected (0.16 sec)

DML COMMANDS:

```
mysql> insert into supplier values(2,'Satvik','8654432318','Bhavani Apartments');
Query OK, 1 row affected (0.19 sec)

mysql> select * from supplier;
+-------------+---------------+---------------------+--------------------+
| supplier_id | supplier_name | contact_info        | address            |
+-------------+---------------+---------------------+--------------------+
|           1 | Sukeerthan    | Haritasa Apartments | Gjarath            |
|           2 | Satvik        | 8654432318          | Bhavani Apartments |
+-------------+---------------+---------------------+--------------------+
2 rows in set (0.08 sec)

mysql> insert into stock values(2,'Raymond',20,1998);
Query OK, 1 row affected (0.04 sec)

mysql> select * from stock;
+----------+--------------+----------+-------+
| stock_id | stock_name   | quantity | price |
+----------+--------------+----------+-------+
|        1 | Ramraj Shirt |       15 |  1499 |
|        2 | Raymond      |       20 |  1998 |
+----------+--------------+----------+-------+
2 rows in set (0.07 sec)

mysql> insert into customer values(2,'Sahana','9876543210','Vishwas Apartments');
Query OK, 1 row affected (0.08 sec)

mysql> select * from customer;
+-------------+---------------+--------------+--------------------+
| customer_id | customer_name | contact_info | address            |
+-------------+---------------+--------------+--------------------+
|           1 | Vinuthna      | 8396364290   | Hasmitha Nandana   |
|           2 | Sahana        | 9876543210   | Vishwas Apartments |
+-------------+---------------+--------------+--------------------+
2 rows in set (0.00 sec)

mysql> insert into purchase values(2,2,2,3,5994);
Query OK, 1 row affected (0.08 sec)
```

```
mysql> select * from purchase;
+-------------+-------------+----------+----------+--------------+
| purchase_id | customer_id | stock_id | quantity | total_amount |
+-------------+-------------+----------+----------+--------------+
|           1 |           1 |        1 |        2 |         2998 |
|           2 |           2 |        2 |        3 |         5994 |
+-------------+-------------+----------+----------+--------------+
2 rows in set (0.06 sec)

mysql>
mysql> insert into bill values(2,2,'2024-05-20',5994);
Query OK, 1 row affected (0.08 sec)

mysql> select * from bill;
+---------+-------------+------------+--------------+
| bill_id | purchase_id | date       | total_amount |
+---------+-------------+------------+--------------+
|       1 |           1 | 2024-05-19 |         2998 |
|       2 |           2 | 2024-05-20 |         5994 |
+---------+-------------+------------+--------------+
2 rows in set (0.00 sec)
```

IMPLEMENTATION :

JAVA-SQL CONNECTIVITY USING JDBC: Java Database Connectivity (JDBC) is an application programming interface (API) for the programming language Java, which defines how a client may access a database. It is a Java-based data access technology used for Java database connectivity. It is part of the Java Standard Edition platform, from Oracle Corporation. It provides methods to query and update data in a database and is oriented towards relational databases.

APPLICATION CODE:

- DatabaseConnection

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
public class DatabaseConnection {
   private static final String url = "jdbc:mysql://localhost:3306/textiles";
   private static final String username = "root";
   private static final String password = "Vinu@2004";
   public static Connection getConnection() throws SQLException {
      return DriverManager.getConnection(url, username, password);
   }
}
```

- MainApp

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
public class MainApp {
  public static void main(String[] args) {
    JFrame frame = new JFrame("Inventory Management");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(400, 300);
    frame.setLayout(new GridLayout(3, 1));
    JButton addStockButton = new JButton("Add Stock");
    JButton billButton = new JButton("Bill");
    JButton searchButton = new JButton("Search");
    addStockButton.addActionListener(new ActionListener() {
      @Override
      public void actionPerformed(ActionEvent e) {
        new AddStockForm().setVisible(true);
      }
    });
      billButton.addActionListener(new ActionListener() {
      @Override
      public void actionPerformed(ActionEvent e) {
        new BillForm().setVisible(true);
      }
    });
    searchButton.addActionListener(new ActionListener() {
      @Override
      public void actionPerformed(ActionEvent e) {
        new SearchForm().setVisible(true);
      }
    });
    frame.add(addStockButton);
```

```
        frame.add(billButton);
        frame.add(searchButton);

        frame.setVisible(true);
    }
}
```

- Stock

```
public class Stock {
    private int stockId;
    private String stockName;
    private int quantity;
    private double price;
    // Getters and Setters
    public int getStockId() {
        return stockId;
    }
    public void setStockId(int stockId) {
        this.stockId = stockId;
    }
    public String getStockName() {
        return stockName;
    }
    public void setStockName(String stockName) {
        this.stockName = stockName;
    }
    public int getQuantity() {
        return quantity;
    }
    public void setQuantity(int quantity) {
        this.quantity = quantity;
    }
```

```
    public double getPrice() {
        return price;
    }
    public void setPrice(double price) {
        this.price = price;
    }
    public String toString() {
        return "Stock ID: " + stockId + ", Item Name: " + stockName + ", Quantity: " + quantity + ", Price: " + price;
    }
}
```

- AddStockForm

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
public class AddStockForm extends JFrame {
    private JTextField stockNameField, stockQuantityField, stockPriceField;
    private JTextField supplierNameField, supplierContactField, supplierAddressField;
    public AddStockForm() {
        setTitle("Add Stock");
        setSize(400, 400);
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        setLayout(new GridLayout(7, 2));
        JLabel stockNameLabel = new JLabel("Stock Name:");
        stockNameField = new JTextField();
        JLabel stockQuantityLabel = new JLabel("Stock Quantity:");
        stockQuantityField = new JTextField();
        JLabel stockPriceLabel = new JLabel("Stock Price:");
        stockPriceField = new JTextField();
        JLabel supplierNameLabel = new JLabel("Supplier Name:");
        supplierNameField = new JTextField();
```

```
JLabel supplierContactLabel = new JLabel("Supplier Contact:");
supplierContactField = new JTextField();
JLabel supplierAddressLabel = new JLabel("Supplier Address:");
supplierAddressField = new JTextField();
JButton addButton = new JButton("Add");
addButton.addActionListener(new ActionListener() {
  @Override
  public void actionPerformed(ActionEvent e) {
    try {
      Supplier supplier = new Supplier();
      supplier.setSupplierName(supplierNameField.getText());
      supplier.setContactInfo(supplierContactField.getText());
      supplier.setAddress(supplierAddressField.getText());
      SupplierDAO supplierDAO = new SupplierDAO();
      supplierDAO.addSupplier(supplier);
      Stock stock = new Stock();
      stock.setStockName(stockNameField.getText());
      stock.setQuantity(Integer.parseInt(stockQuantityField.getText()));
      stock.setPrice(Double.parseDouble(stockPriceField.getText()));
      StockDAO stockDAO = new StockDAO();
      stockDAO.addStock(stock);
      JOptionPane.showMessageDialog(null, "Stock and Supplier added successfully!");
    } catch (Exception ex) {
      JOptionPane.showMessageDialog(null, "Error: " + ex.getMessage());
    }
  }
});
add(stockNameLabel);
add(stockNameField);
add(stockQuantityLabel);
add(stockQuantityField);
add(stockPriceLabel);
```

```
        add(stockPriceField);
        add(supplierNameLabel);
        add(supplierNameField);
        add(supplierContactLabel);
        add(supplierContactField);
        add(supplierAddressLabel);
        add(supplierAddressField);
        add(new JLabel()); // Empty label for spacing
        add(addButton);
    }
}
```

- StockDAO

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.ArrayList;
import java.util.List;
public class StockDAO {
    public List<Stock> searchStocks(Connection connection, String searchText) {
        List<Stock> stocks = new ArrayList<>();
        String query = "SELECT * FROM stock WHERE item_name LIKE ?";
        try (PreparedStatement statement = connection.prepareStatement(query)) {
            statement.setString(1, "%" + searchText + "%");
            ResultSet resultSet = statement.executeQuery();
            while (resultSet.next()) {
                Stock stock = new Stock();
                stock.setStockId(resultSet.getInt("stock_id"));
                stock.setStockName(resultSet.getString("item_name"));
                stock.setQuantity(resultSet.getInt("quantity"));
                stock.setPrice(resultSet.getDouble("price"));
                stocks.add(stock);
```

```
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    return stocks;
  }
  public boolean addStock(Stock stock) {
    String query = "INSERT INTO stock (item_name, quantity, price) VALUES (?, ?, ?)";
    try (Connection connection = DatabaseConnection.getConnection();
         PreparedStatement statement = connection.prepareStatement(query)) {
        statement.setString(1, stock.getStockName());
        statement.setInt(2, stock.getQuantity());
        statement.setDouble(3, stock.getPrice());
        int rowsInserted = statement.executeUpdate();
        return rowsInserted > 0;
    } catch (Exception e) {
        e.printStackTrace();
        return false;
    }
  }
}
```

- Bill

```
public class Bill {
  private int billId;
  private int purchaseId;
  private double totalAmount;
  public int getBillId() {
    return billId;
  }
  public void setBillId(int billId) {
    this.billId = billId;
  }
```

```
    public int getPurchaseId() {
        return purchaseId;
    }
    public void setPurchaseId(int purchaseId) {
        this.purchaseId = purchaseId;
    }
    public double getTotalAmount() {
        return totalAmount;
    }
    public void setTotalAmount(double totalAmount) {
        this.totalAmount = totalAmount;
    }
    @Override
    public String toString() {
        return "Bill{" +
                "billId=" + billId +
                ", purchaseId=" + purchaseId +
                ", totalAmount=" + totalAmount +
                '}';
    }
}
```

- BillForm

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
public class BillForm extends JFrame {
    private JTextField customerNameField;
    private JTextField contactInfoField;
    private JTextField addressField;
    private JTextField stockIdField;
```

```
private JTextField quantityField;
private JTextField totalAmountField;
private JTextArea billDetailsArea;
private BillingService billingService = new BillingService();
public BillForm() {
  setTitle("Bill");
  setSize(400, 400);
  setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
  setLayout(new BorderLayout());
  customerNameField = new JTextField(20);
  contactInfoField = new JTextField(20);
  addressField = new JTextField(20);
  stockIdField = new JTextField(20);
  quantityField = new JTextField(20);
  totalAmountField = new JTextField(20);
  billDetailsArea = new JTextArea();
  billDetailsArea.setEditable(false);
  JPanel panel = new JPanel(new GridLayout(7, 2));
  panel.add(new JLabel("Customer Name:"));
  panel.add(customerNameField);
  panel.add(new JLabel("Contact Info:"));
  panel.add(contactInfoField);
  panel.add(new JLabel("Address:"));
  panel.add(addressField);
  panel.add(new JLabel("Stock ID:"));
  panel.add(stockIdField);
  panel.add(new JLabel("Quantity:"));
  panel.add(quantityField);
  panel.add(new JLabel("Total Amount:"));
  panel.add(totalAmountField);
  JButton generateBillButton = new JButton("Generate Bill");
  panel.add(new JLabel()); // Empty label for spacing
```

```
    panel.add(generateBillButton);
    add(panel, BorderLayout.NORTH);
    add(new JScrollPane(billDetailsArea), BorderLayout.CENTER);
    generateBillButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            generateBill();
        }
    });
}
private void generateBill() {
    try {
        String customerName = customerNameField.getText();
        String contactInfo = contactInfoField.getText();
        String address = addressField.getText();
        int stockId = Integer.parseInt(stockIdField.getText());
        int quantity = Integer.parseInt(quantityField.getText());
        double totalAmount = Double.parseDouble(totalAmountField.getText());
        Customer customer = new Customer();
        customer.setCustomerName(customerName);
        customer.setContactInfo(contactInfo);
        customer.setAddress(address);
        Purchase purchase = new Purchase();
        purchase.setStockId(stockId);
        purchase.setQuantity(quantity);
        purchase.setTotalAmount(totalAmount);
        Bill bill = new Bill();
        bill.setTotalAmount(totalAmount);
        billingService.generateBill(customer, purchase, bill);
        billDetailsArea.setText("");
        billDetailsArea.append("Customer Name: " + customerName + "\n");
        billDetailsArea.append("Contact Info: " + contactInfo + "\n");
```

```
            billDetailsArea.append("Address:  " + address + "\n");
            billDetailsArea.append("Stock  ID: " + stockId + "\n");
            billDetailsArea.append("Quantity:  " + quantity + "\n");
            billDetailsArea.append("Total  Amount: " + totalAmount  + "\n");
            billDetailsArea.append("Bill  ID: " + bill.getBillId()  + "\n");
        } catch (NumberFormatException  ex) {
            JOptionPane.showMessageDialog(this,  "Invalid input: " + ex.getMessage());
        }
    }
}
```

- BillDAO

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
public class BillDAO {
    public void addBill(Connection  connection, Bill bill) throws SQLException {
        String query = "INSERT INTO bill (purchase_id, total_amount)  VALUES (?, ?)";
        try (PreparedStatement statement = connection.prepareStatement(query,
                PreparedStatement.RETURN_GENERATED_KEYS))  {
            statement.setInt(1, bill.getPurchaseId());
            statement.setDouble(2, bill.getTotalAmount());
            int rowsInserted = statement.executeUpdate();
            if (rowsInserted > 0) {
                try (ResultSet generatedKeys = statement.getGeneratedKeys())  {
                    if (generatedKeys.next()) {
                        bill.setBillId(generatedKeys.getInt(1));
                    }
                }
            }
        }
```

```
    }
    public Bill getBillById(Connection connection, int billId) throws SQLException {
        String query = "SELECT * FROM bill WHERE bill_id = ?";
        try (PreparedStatement statement = connection.prepareStatement(query)) {
            statement.setInt(1, billId);
            try (ResultSet resultSet = statement.executeQuery()) {
                if (resultSet.next()) {
                    Bill bill = new Bill();
                    bill.setBillId(resultSet.getInt("bill_id"));
                    bill.setPurchaseId(resultSet.getInt("purchase_id"));
                    bill.setTotalAmount(resultSet.getDouble("total_amount"));
                    return bill;
                } else {
                    return null;
                }
            }
        }
    }
}
```

- BillingService

```
import java.sql.Connection;
import java.sql.SQLException;
public class BillingService {
    private CustomerDAO customerDAO;
    private PurchaseDAO purchaseDAO;
    private BillDAO billDAO;
    public BillingService() {
        this.customerDAO = new CustomerDAO();
        this.purchaseDAO = new PurchaseDAO();
        this.billDAO = new BillDAO();
    }
    public void generateBill(Customer customer, Purchase purchase, Bill bill) {
```

```
        try (Connection connection = DatabaseConnection.getConnection()) {
            connection.setAutoCommit(false);
            try {
                customerDAO.addCustomer(connection, customer);
                purchase.setCustomerId(customer.getCustomerId());
                purchaseDAO.addPurchase(connection, purchase);
                bill.setPurchaseId(purchase.getPurchaseId());
                billDAO.addBill(connection, bill);
                connection.commit();
            } catch (SQLException e) {
                connection.rollback();
                throw new RuntimeException("Error generating bill", e);
            }
        } catch (SQLException e) {
            throw new RuntimeException("Error connecting to database", e);
        }
    }
    public Bill searchBillById(int billId) {
        try (Connection connection = DatabaseConnection.getConnection()) {
            return billDAO.getBillById(connection, billId);
        } catch (SQLException e) {
            throw new RuntimeException("Error searching for bill", e);
        }
    }
}
```

- SearchType

```
public enum SearchType {
    STOCK("Stock"),
    CUSTOMER("Customer"),
    PURCHASE("Purchase");
    private String displayName;
    SearchType(String displayName) {
```

```
        this.displayName = displayName;
    }
    @Override
    public String toString() {
        return displayName;
    }
}
```

- SearchForm

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
public class SearchForm extends JFrame {
    private JTextField searchField;
    private JTextArea searchResultsArea;
    private BillingService billingService = new BillingService();
    public SearchForm() {
        setTitle("Search Bills");
        setSize(400, 400);
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        setLayout(new BorderLayout());
        searchField = new JTextField(20);
        searchResultsArea = new JTextArea();
        searchResultsArea.setEditable(false);
        JPanel panel = new JPanel(new GridLayout(2, 1));
        panel.add(new JLabel("Enter Bill ID:"));
        panel.add(searchField);
        JButton searchButton = new JButton("Search");
        panel.add(new JLabel()); // Empty label for spacing
        panel.add(searchButton);
        add(panel, BorderLayout.NORTH);
```

```
        add(new JScrollPane(searchResultsArea), BorderLayout.CENTER);
        searchButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                searchBill();
            }
        });
    }
    private void searchBill() {
        try {
            int billId = Integer.parseInt(searchField.getText());

            Bill bill = billingService.searchBillById(billId);
            if (bill != null) {
                searchResultsArea.setText("");
                searchResultsArea.append("Bill ID: " + bill.getBillId() + "\n");
                searchResultsArea.append("Purchase ID: " + bill.getPurchaseId() + "\n");
                searchResultsArea.append("Total Amount: " + bill.getTotalAmount() + "\n");
            } else {
                searchResultsArea.setText("Bill not found.");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(this, "Invalid input: " + ex.getMessage());
        }
    }
}
```

- SearchDAO

```
import java.sql.*;
import java.util.ArrayList;
import java.util.List;
public class SearchDAO {
    public List<Stock> searchStocks(String keyword) throws SQLException {
```

```
    List<Stock> stocks = new ArrayList<>();
    String query = "SELECT * FROM stock WHERE stock_name LIKE ? OR stock_id LIKE ?";
    try (Connection connection = DatabaseConnection.getConnection();
         PreparedStatement statement = connection.prepareStatement(query)) {
      statement.setString(1, "%" + keyword + "%");
      statement.setString(2, "%" + keyword + "%");
      ResultSet resultSet = statement.executeQuery();
      while (resultSet.next()) {
        Stock stock = new Stock();
        stock.setStockId(resultSet.getInt("stock_id"));
        stock.setStockName(resultSet.getString("stock_name"));
        stock.setQuantity(resultSet.getInt("quantity"));
        stock.setPrice(resultSet.getDouble("price"));
        stocks.add(stock);
      }
    }
    return stocks;
  }
  public List<Customer> searchCustomers(String keyword) throws SQLException {
    List<Customer> customers = new ArrayList<>();
    String query = "SELECT * FROM customer WHERE customer_name LIKE ? OR customer_id LIKE ?";
    try (Connection connection = DatabaseConnection.getConnection();
         PreparedStatement statement = connection.prepareStatement(query)) {
      statement.setString(1, "%" + keyword + "%");
      statement.setString(2, "%" + keyword + "%");
      ResultSet resultSet = statement.executeQuery();
      while (resultSet.next()) {
        Customer customer = new Customer();
        customer.setCustomerId(resultSet.getInt("customer_id"));
        customer.setCustomerName(resultSet.getString("customer_name"));
customer.setContactInfo(resultSet.getString("contact_info"));
        customer.setAddress(resultSet.getString("address"));
```

```
            customers.add(customer);
        }
    }
    return customers;
}

public List<Purchase> searchPurchases(String keyword) throws SQLException {
    List<Purchase> purchases = new ArrayList<>();
    String query = "SELECT * FROM purchase WHERE purchase_id LIKE ? OR customer_id LIKE ?";
    try (Connection connection = DatabaseConnection.getConnection();
         PreparedStatement statement = connection.prepareStatement(query)) {
        statement.setString(1, "%" + keyword + "%");
        statement.setString(2, "%" + keyword + "%");
        ResultSet resultSet = statement.executeQuery();
        while (resultSet.next()) {
            Purchase purchase = new Purchase();
            purchase.setPurchaseId(resultSet.getInt("purchase_id"));
            purchase.setCustomerId(resultSet.getInt("customer_id"));
            purchase.setStockId(resultSet.getInt("stock_id"));
            purchase.setQuantity(resultSet.getInt("quantity"));
purchase.setTotalAmount(resultSet.getDouble("total_amount"));
            purchases.add(purchase);
        }
    }
    return purchases;
}
}
```

- Customer

```
public class Customer {
    private int customerId;
    private String customerName;
    private String contactInfo;
```

```
    private String address;
    // Getters and Setters
    public int getCustomerId() {
        return customerId;
    }
    public void setCustomerId(int customerId) {
        this.customerId = customerId;
    }
    public String getCustomerName() {
        return customerName;
    }
    public void setCustomerName(String customerName) {
        this.customerName = customerName;
    }
    public String getContactInfo() {
        return contactInfo;
    }
    public void setContactInfo(String contactInfo) {
        this.contactInfo = contactInfo;
    }
    public String getAddress() {
        return address;
    }
    public void setAddress(String address) {
        this.address = address;
    }
    public String toString() {
        return "Customer ID: " + customerId + ", Name: " + customerName + ", Phone: " + contactInfo
                + ", Address: " + address;
    }
}
```

- CustomerDAO

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;
public class CustomerDAO {
  public int addCustomer(Connection connection, Customer customer) throws SQLException {
    String query = "INSERT INTO customer (customer_name, contact_info, address) VALUES (?, ?, ?)";
    try (PreparedStatement statement = connection.prepareStatement(query,
        PreparedStatement.RETURN_GENERATED_KEYS)) {
      statement.setString(1, customer.getCustomerName());
      statement.setString(2, customer.getContactInfo());
      statement.setString(3, customer.getAddress());
      int rowsInserted = statement.executeUpdate();
      if (rowsInserted > 0) {
        try (ResultSet generatedKeys = statement.getGeneratedKeys()) {
          if (generatedKeys.next()) {
            return generatedKeys.getInt(1);
          }
        }
      }
    }
    return -1; // Return -1 or any indicator of failure to insert the customer
  }
  public List<Customer> searchCustomers(Connection connection, String searchText) throws SQLException
{
    List<Customer> customers = new ArrayList<>();
    String query = "SELECT * FROM customer WHERE customer_name LIKE ? OR contact_info LIKE ?";
    try (PreparedStatement statement = connection.prepareStatement(query)) {
      statement.setString(1, "%" + searchText + "%");
      statement.setString(2, "%" + searchText + "%");
```

```
        ResultSet resultSet = statement.executeQuery();
        while (resultSet.next()) {
          Customer customer = new Customer();
          customer.setCustomerId(resultSet.getInt("customer_id"));
          customer.setCustomerName(resultSet.getString("customer_name"));
customer.setContactInfo(resultSet.getString("contact_info"));
          customer.setAddress(resultSet.getString("address"));
          customers.add(customer);
        }
      }
      return customers;
  }
}
```

- Purchase

```
public class Purchase {
  private int purchaseId;
  private int customerId;
  private int stockId;
  private int quantity;
  private double totalAmount;
  // Getters and Setters
  public int getPurchaseId() {
    return purchaseId;
  }
  public void setPurchaseId(int purchaseId) {
    this.purchaseId = purchaseId;
  }
  public int getCustomerId() {
    return customerId;
  }
  public void setCustomerId(int customerId) {
```

```
    this.customerId = customerId;
  }
  public int getStockId() {
    return stockId;
  }
  public void setStockId(int  stockId) {
    this.stockId = stockId;
  }
  public int getQuantity() {
    return quantity;
  }
  public void setQuantity(int  quantity) {
    this.quantity = quantity;
  }
  public double getTotalAmount() {
    return totalAmount;
  }
   public void setTotalAmount(double  totalAmount) {
    this.totalAmount = totalAmount;
  }
  public String toString() {
    return "Purchase ID: " + purchaseId + ", Customer ID: " + customerId + ", Stock ID: " + stockId + ",
Quantity: "
        + quantity;
  }
}
```

- PurchaseDAO

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
```

```
import java.util.List;
public class PurchaseDAO {
  public int addPurchase(Connection connection, Purchase purchase) throws SQLException {
    String query = "INSERT INTO purchase (customer_id, stock_id, quantity, total_amount) VALUES
(?, ?, ?, ?)";
    try (PreparedStatement statement = connection.prepareStatement(query,
        PreparedStatement.RETURN_GENERATED_KEYS)) {
      statement.setInt(1, purchase.getCustomerId());
      statement.setInt(2, purchase.getStockId());
      statement.setInt(3, purchase.getQuantity());
      statement.setDouble(4, purchase.getTotalAmount());
    int rowsInserted = statement.executeUpdate();
      if (rowsInserted > 0) {
        try (ResultSet generatedKeys = statement.getGeneratedKeys()) {
          if (generatedKeys.next()) {
            return generatedKeys.getInt(1);
          }
        }
      }
    }
    return -1; // Return -1 or any indicator of failure to insert the purchase
  }
  public List<Purchase> searchPurchases(Connection connection, String searchText) throws SQLException {
    List<Purchase> purchases = new ArrayList<>();
    String query = "SELECT * FROM purchase WHERE purchase_id LIKE ? OR customer_id LIKE ?";
    try (PreparedStatement statement = connection.prepareStatement(query)) {
      statement.setString(1, "%" + searchText + "%");
      statement.setString(2, "%" + searchText + "%");
      ResultSet resultSet = statement.executeQuery();
      while (resultSet.next()) {
        Purchase purchase = new Purchase();
        purchase.setPurchaseId(resultSet.getInt("purchase_id"));
```

```
            purchase.setCustomerId(resultSet.getInt("customer_id"));
            purchase.setStockId(resultSet.getInt("stock_id"));
            purchase.setQuantity(resultSet.getInt("quantity"));
purchase.setTotalAmount(resultSet.getDouble("total_amount"));
            purchases.add(purchase);
        }
      }
      return purchases;
   }
}
```

- Supplier

```
public class Supplier {
   private int supplierId;
   private String supplierName;
   private String contactInfo;
   private String address;
   // Getters and Setters
   public int getSupplierId() {
      return supplierId;
   }
   public void setSupplierId(int supplierId) {
      this.supplierId = supplierId;
   }
   public String getSupplierName() {
      return supplierName;
   }
   public void setSupplierName(String supplierName) {
      this.supplierName = supplierName;
   }
   public String getContactInfo() {
      return contactInfo;
   }
```

```
    public void setContactInfo(String contactInfo) {
        this.contactInfo = contactInfo;
    }
    public String getAddress() {
        return address;
    }
    public void setAddress(String address) {
        this.address = address;
    }
}
```

- SupplierDAO

```
import java.sql.*;
import java.util.ArrayList;
import java.util.List;
public class SupplierDAO {
    public void addSupplier(Supplier supplier) throws SQLException {
        String query = "INSERT INTO supplier (supplier_name, contact_info, address) VALUES (?, ?, ?)";
        try (Connection connection = DatabaseConnection.getConnection();
             PreparedStatement statement = connection.prepareStatement(query)) {
            statement.setString(1, supplier.getSupplierName());
            statement.setString(2, supplier.getContactInfo());
            statement.setString(3, supplier.getAddress());
            statement.executeUpdate();
        }
    }
    public List<Supplier> getAllSuppliers() throws SQLException {
        List<Supplier> suppliers = new ArrayList<>();
        String query = "SELECT * FROM supplier";
        try (Connection connection = DatabaseConnection.getConnection();
             Statement statement = connection.createStatement();
             ResultSet resultSet = statement.executeQuery(query)) {
            while (resultSet.next()) {
```

```
            Supplier supplier = new Supplier();
            supplier.setSupplierId(resultSet.getInt("supplier_id"));
            supplier.setSupplierName(resultSet.getString("supplier_name"));
            supplier.setContactInfo(resultSet.getString("contact_info"));
            supplier.setAddress(resultSet.getString("address"));
            suppliers.add(supplier);
        }
    }
    return suppliers;
  }
}
```

RESULTS:

I have successfully completed the mini project "Textiles Store Management System".

DISCUSSION AND FUTURE WORK:

The Textiles Store Management System (TSMS) is an innovative solution designed to optimize the operations of a textile retail store by integrating key functions such as inventory management, customer relations, and supplier interactions. The system enhances efficiency and accuracy by automating routine tasks, reducing human error, and ensuring real-time updates on stock levels, which helps maintain an optimal inventory. The customer management feature allows for efficient handling of customer data and purchase history, facilitating personalized service and targeted marketing efforts. The system's design also supports detailed tracking of products and supplier information, which is crucial for avoiding stockouts and managing supplier relationships effectively. However, the scalability of the system needs to be tested for larger stores, and future enhancements could include optimizing database queries and indexing frequently accessed data to handle higher transaction volumes.

Ensuring the security and integrity of data is paramount, particularly for sensitive customer and financial information. Implementing role-based access control and conducting regular audits are essential for maintaining data security. Future work could focus on integrating advanced security protocols and encryption methods. Additionally, the system could benefit from integration with e-commerce platforms to expand the store's reach and synchronize inventory between physical and online stores. Implementing advanced data analytics and reporting tools would provide deeper insights into sales trends and customer preferences, while machine learning algorithms could predict demand and optimize inventory levels. Developing a mobile application for the TSMS would enhance flexibility for

employees and provide customers with features like loyalty programs and personalized offers.

Further enhancements could include automated restocking systems that communicate directly with suppliers, improving inventory management and reducing manual intervention. Enhancing the CRM features with automated follow-ups and personalized marketing campaigns would boost customer engagement and loyalty. Integrating blockchain technology could improve supply chain transparency and ensure the authenticity and ethical sourcing of textiles. Additionally, incorporating IoT devices for real-time inventory tracking and smart shelves could further enhance efficiency and customer experience. By addressing these future directions, the TSMS can evolve into a more comprehensive and intelligent platform, providing greater value to both the store and its customers.

REFERENCES

- https://docs.oracle.com/javase/7/docs/api/
- https://www.javatpoint.com/java-swing
- https://stackoverflow.com/