**CS361 Software Engineering**

Retail Inventory Management System

Group 2
Christian Young, Dylan Finely, Jared Svoboda,
Georges Nchouwat, Zilun Du

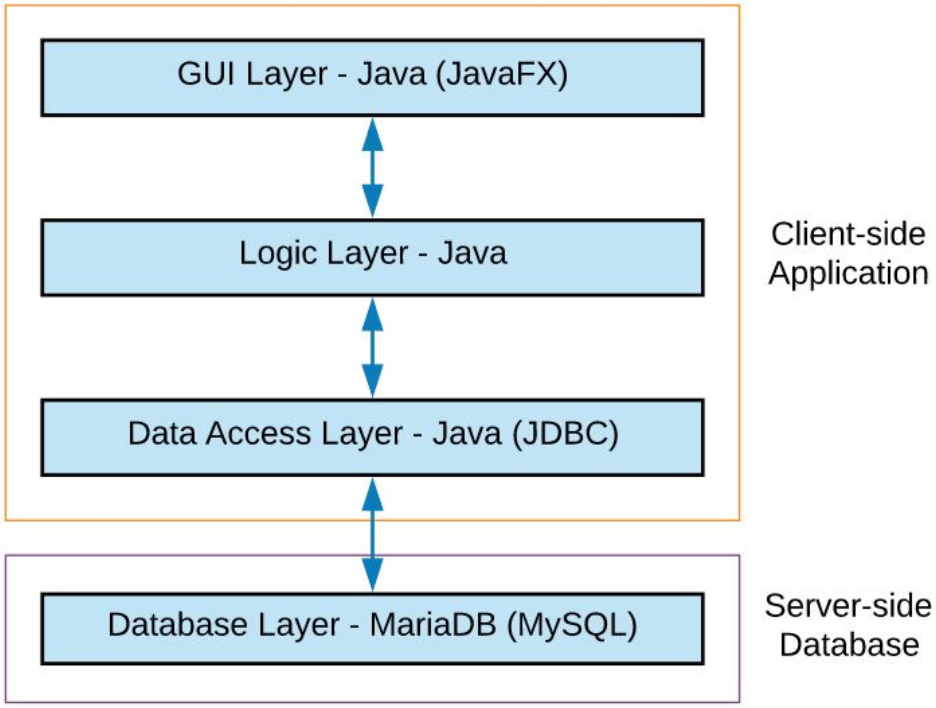
# Design Document

# 1. Introduction

The purpose of this design is to demonstrate the high level architecture and entity relations, for the Retail Inventory Management System. This document will include architecture and entity relation diagrams showing the relationship between different parts of the system, as well as the relationships between tables in the database. The audience of this document is software engineers and system architects who will be implementing and maintaining the described project.

# 2. Architecture

## 2.1 Introduction




A layered model describes the high-level architectural design. The lowest level (the Database Layer) will be a MariaDB database that stores information. Above that is the Data Access Layer, which uses the Java JDBC framework to access and manipulate information contained within the database. The Logic Layer, also written in Java, will be the heart of the client-side application. It will get data from the Data Access Layer and perform the program's various operations on it. The top layer of this model is the GUI Layer, written using JavaFX, which allows users to easily interact with and view data from the system's Logic Layer through its interface.

## 2.2 Modules

### 2.2.1 Database Layer

The database layer is responsible for the persistence of data within the system and establishing the relationships between data and the tables it is held within. The database system will be implemented using MariaDB and MySQL. Relationships between data and subsequent tables will be further elaborated on below.

### 2.2.2 Data Access Layer

The data access layer is the module of the client-side Java program which is responsible for accessing data from the database and transforming it as needed so that the logic layer can work with it. The data access layer also implements changes to the database whenever the logic layer commands it to.

### 2.2.3 Logic Layer

The data access layer is the module of the client-side Java program which is responsible for performing all necessary functions on the Java objects in the system. Through interaction with the GUI layer the module will display desired information back to the GUI layer. The logic layer is also responsible for communication between the GUI layer and Data Access Layer so that changes made to data by the user at the GUI level are sent and eventually enacted at the Database Layer, where they are stored permanently .

### 2.2.4 GUI Layer

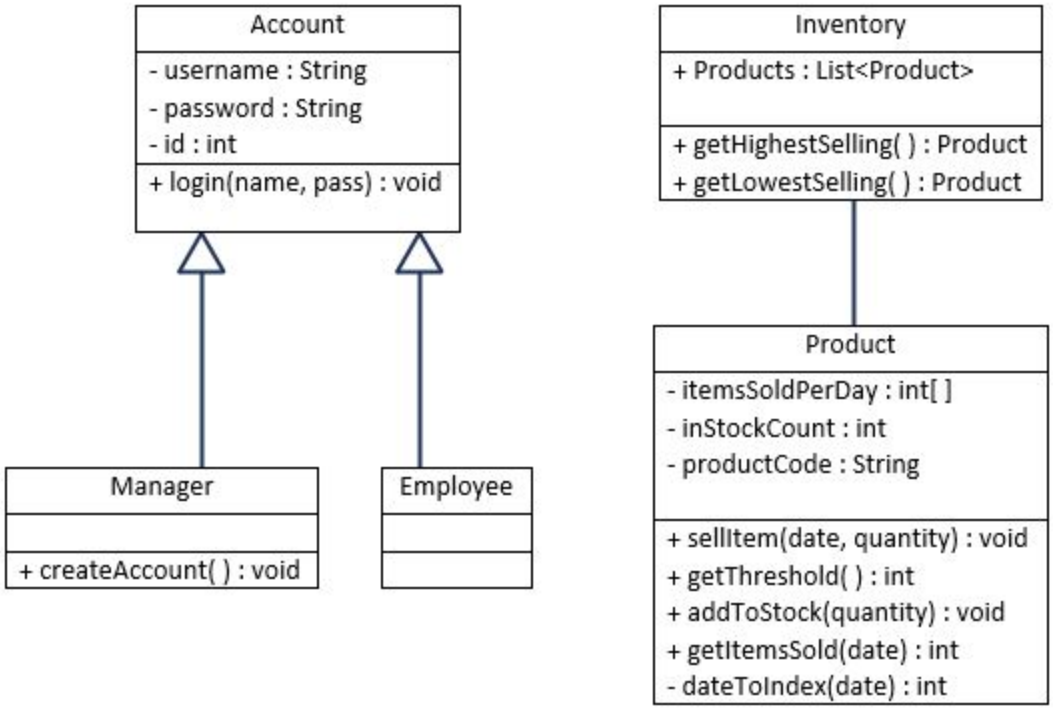
The GUI layer is the module of the client-side Java program which generates the user interface, and is the primary layer that the user interacts with. Through this layer, a user can log into an account and view representations of the database information processed in the logic layer. In earlier phases this will be emulated with the console and later developed.

# 3. Class Diagrams (Client-side application)

## 3.1 Logic Layer Classes

The logic layer of the system will be using Java object to interact and manipulate data. This includes accounts of employees and managers, inventory, and products. The figure below shows the structure of these objects and their functions.

### 3.1.1 Schema




*Class Diagram*

### 3.1.2 Schema Information

**Account:** This class will be implemented in Java. An *account* class will be an abstract class with variables for *username, password,* and *id* (ID). An *account* will have a *login* method that will require a username and password.

**Employee:** This class will be a subclass of *account.* It will not have any further members or methods than the basic *account*.

**Manager:** This class will be a subclass of *account.* It will not have any further members than the basic account. It will, however, have an additional method to create a new account.

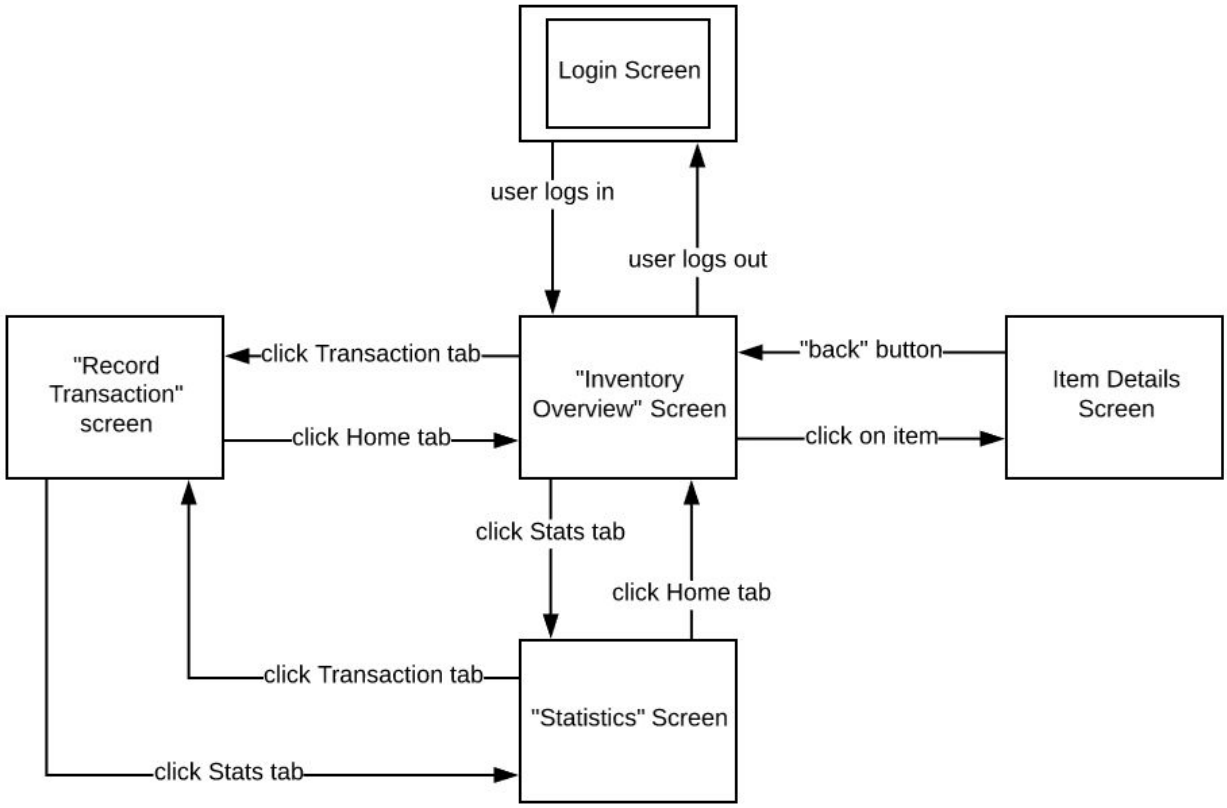
**Product:** This class will have variables for *itemsSoldPerDay, inStockCount,* and *productCode.* There will also be methods to *sellItem, getThreshold, addToStock, getItemsSold,* and convert *dateToIndex.*

**Inventory:** This class will have a list of all the *products.* There will also be methods to *getHighestSelling* and *getLowestSelling.*

## 3.2 GUI Layer Flow/Organization

In earlier phases this will be emulated with the console and later developed.

### 3.2.1 Schema




*GUI Flow diagram*

### 3.1.2 Schema Information

The GUI will be contained in five distinct “screens”, in which related actions and information are visually organized.

**Login Screen:** From here the user can enter the username and password of an existing employee or manager account into two respective text-entry fields in order to log in and access the rest of the application. There will also be a button labelled “create new account” which, when clicked, will reveal an additional text-entry field. In this state, the original text fields now accept a username and password for the account to be created, and the newly revealed text fields will

require an existing manager's password to verify the account creation. If the password entered in the new text fields is not associated with an existing Manager account, then the account creation will fail. This ensures that only Managers have the ability to generate new accounts for employees and other managers. If account creation is successful, "account created" will be displayed and the login screen will return to its original state.
**Inventory Overview Screen:** This screen is the first to display after logging in. For Manager accounts, a header at the top of this screen will show three labelled tabs. Clicking on one of these three tabs instantly navigates the user to the Record Transaction Screen, Statistics Screen, or Inventory Overview screen from any of the other two. For employee accounts, the tab representing the Statistics screen will be absent.

The main portion of the Inventory Overview screen will display a list representing the store's inventory, with products as the list elements. Some product information such as name and in-stock count will be displayed in rows, where each row represents one product.

Clicking on the row of a product brings up a window which gives more detailed information on that product - this window is the **Item Details Screen**. This screen is a smaller, detached window appearing in front of the rest of the GUI. It can be moved around by clicking and dragging, and closed by pressing a "back" or "close" button. Several instances can be open at once, to show details of several items. Manager accounts will have the ability to edit information displayed in this window, such as price, product code, name, etc. Such edits, once made, are reflected in the database.

**Record Transaction Screen:** This is where a user can update the quantity of products in stock by recording either outgoing sales of products or incoming shipments of them.

**Statistics Screen:** This screen is only accessible to manager accounts. It allows managers to view aggregate information on outgoing sales, incoming products, monetary value of sales, etc. over a span of time up to a year back.

### 3.3 Data Access Layer

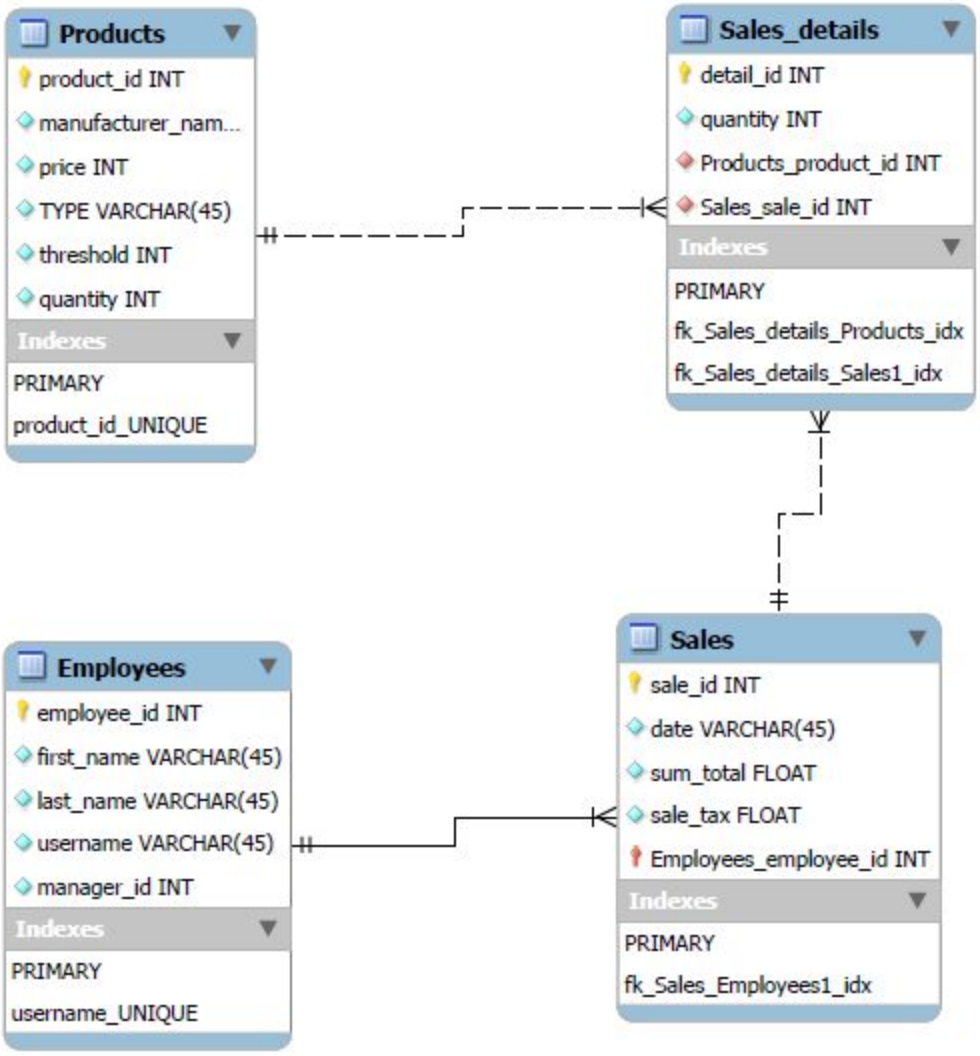
Java Database Connectivity (JDBC) will be used to link the business logic tier to the database. This will allow for updating, retrieving, creating, and deleting data stored in the database.

## 4. Database Contents (Server-side Database)

The system will be using an MariaDB database to hold all the persistent data necessary. This includes information on products, sales, and employees and their relationship to one another. The

figure below shows each table in the database with its respective columns as well as the relationships between the different tables.

## 4.1 Schema




## 4.2 Schema Information

The data in the database shall be organized in the following tables and columns:

**Products:** This table holds data for all products in stock and their details. Each product is to have an id as a key with additional information such as the manufacturer, price, its type, stock threshold, and quantity in the other columns. It has an one-to-many relation with sales_details table which will help update the inventory.

**Sales Details:** This table helps manage the many-to-many relation between the tables Products and Sales. Thus it has many-to-one relationship product table and sales table; it holds two foreign keys and data related to sales made such as id and the product sold.

**Sales:** This table stores all sales made by an employee. Each sale has a primary key of an id with the other columns storing data on the sale's date, total, employee, and tax. In order to keep track of which employee made the sale, this table has a many-to-one relationship with employee table. With sales_details table, it has an one-to-many relationship.

**Employees:** This table holds the data for all employee. Each employee is to have an id as a primary key with an employee's first name, last name, username, and manager status in subsequent column. It has an one-to-many relationship with sales table.