**Institute of Technology, Carlow**

**Report**

**for**

**Customer Invoice System**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Name:** Huu Chi Huynh

**Date:** 08/04/2022

**Tutor:** Jason Barron

# 

# Table of Contents

Section Page

Description 3

Requirements 5

Database tables 6

ER Diagram 9

Interesting source code snippets 10

Tests 16

## Description:

In this project, I had to make a **Customer Invoice System** in **Java** using a minimum of 3 DB tables.

I had to implement a lot of classes to make this system:

* Account
* AccountType
* Company
* DatabaseHandler
* Invoice
* InvoiceItem
* Item
* Main
* Packet
* PacketResult
* Product
* UnEditableDefaultTableModel

And I had to make a bunch of JFrame forms from **Netbeans** to handle the front-end:

* CompanyMenu
* LoginAccountForm
* SelectAccountType
* SignUpAccountForm

And I used a lot of different software throughout this project:

* **Netbeans** as my IDE as it has a powerful GUI builder
* **Docker** as a container for the **mySQL** database
* **MySQL Workbench** to view the database tables
* **Microsoft Word** for documentation writing
* **Github** and **Github desktop** for my repository
* **OBS studio** to record my screencast
* **StarUML** to create my ER diagram
* **VSCode** to grab code snippets
* **Snippping Tool** from Windowsto take DB screenshots

And lastly this report was commissioned by Jason Barron in the Object Orientated Software Development module in IT Carlow.

The document contains the following sections:

* Description
* Requirements
* Database tables
* ER Diagram
* Interesting source code snippets
* Tests

## Requirements:

A Customer Invoice Management system is a system that can be used to manage customer invoices. The customers can login and view their invoices and so as the companies as well. The companies which use the system can create invoices which they can hand out to customers.

Both companies and customers can sign up to the system.

Products can also be added into the system with a quantity and cost defined by the company.

Companies should be able to view basic information for the customers.

Invoices must contain the customer information and the products that company has added to the invoice.

The information of the product or invoice can be changed later by the company which created the item.

All products are owned by individual companies and should not be visible by other companies.

## Database tables:

## **Account:**

## 

## **Company:**

## 

## **Customer:**

## 

## **Invoice:**

## 

## **Invoice Item:**

## 

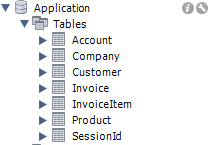
## **Product:**

## 

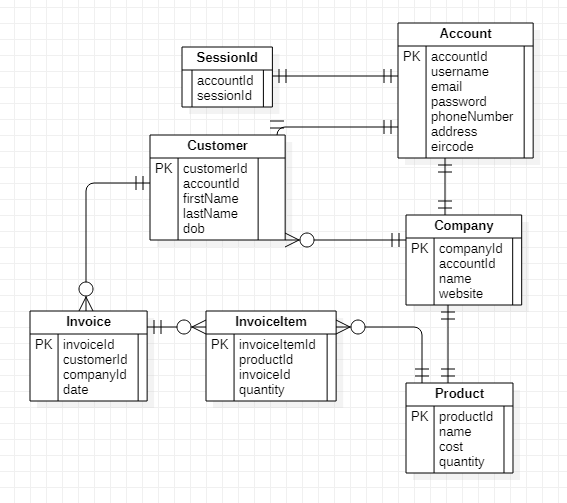
## **Session Id:**

## 

## **All Tables:**



## ER Diagram:



## Interesting source code snippets:

The following function is interesting as it handles most of the login of the user, checking the username and password and creating the session id if the user has successfully logged in.

Used as soon as the user is logging in.

|  |
| --- |
| /**\*\***  **\* Checks both the username and password inputted**  **\* If the username and password are found then**  **\* Create / Update the session id and upload it into the database**  **\* @param username**  **\* @param password**  **\* @return (PacketResult.SUCCESS and sessionId) if successful**  **\*/**  **private static Packet requestLogin(String username, String password) {**  **Packet accountIdPacket = getAccountId(username); // Does username exist**  **if (accountIdPacket.getResult() != PacketResult.SUCCESS) { // Check if packet was successful**  **return accountIdPacket;**  **}**  **int accountId = (int) accountIdPacket.getInformation();**  **if (accountId > 0) { // Verify if account username exists**  **Packet canLoginPacket = canLogin(username, password); // Attempt login with username and password**  **if (canLoginPacket.getResult() == PacketResult.SUCCESS) {**  **String sessionId = UUID.randomUUID().toString(); // Create session id**  **DatabaseHandler handler = DatabaseHandler.getInstance();**  **if (handler.isConnected()) {**  **Object[] info = {username, password};**  **// Attempt to get session id from username and password**  **List<List<Object>> result = handler.get("Application.SessionId.sessionId FROM Application.SessionId JOIN Application.Account ON Application.Account.accountId = Application.SessionId.accountId AND Application.Account.username = ? AND Application.Account.password = ?", info, 1);**  **if (result.isEmpty()) { // No session id exists**  **if (handler.isConnected()) {**  **Object[] args = {accountId, sessionId};**  **boolean success = handler.insert("SessionId(accountId,sessionId) VALUES (?,?)", args);**  **// Create session id**  **if (success) {**  **return new Packet(PacketResult.SUCCESS, sessionId);**  **} else {**  **return new Packet(PacketResult.DATABASE\_ERROR);**  **}**  **} else {**  **return new Packet(PacketResult.CONNECTION\_ERROR);**  **}**  **} else { // Session id already exists**  **Object[] args = {sessionId, accountId};**  **if (handler.isConnected()) {**  **// Update session id**  **boolean success = handler.update("SessionId SET sessionId=? WHERE accountId=?", args);**  **if (success) {**  **return new Packet(PacketResult.SUCCESS, sessionId);**  **} else {**  **return new Packet(PacketResult.DATABASE\_ERROR);**  **}**  **} else {**  **return new Packet(PacketResult.CONNECTION\_ERROR);**  **}**  **}**  **} else {**  **return new Packet(PacketResult.CONNECTION\_ERROR);**  **}**  **} else {**  **return canLoginPacket;**  **}**  **}**  **return new Packet(PacketResult.ERROR\_OCCURRED); // No account exists??? (This should never happen)**  **}** |

The following function is interesting as it handles the session id checking of the user and is called in every login action

Checking if the session id and account id exists inside of the session id DB

|  |
| --- |
| **/\*\***  **\* \***  **\* Checks if account id and session id matches inside of the SessionId**  **\* database table. [USED TO CHECK IF USER IS SIGNED IN OR ACCOUNT EXISTS]**  **\***  **\* @param accountId Account Id of user**  **\* @param sessionId Session Id of current session**  **\* @return (PacketResult.SUCCESS) if the session id exists**  **\*/**  **private static Packet checkSessionId(int accountId, String sessionId) {**  **Object[] info = {accountId, sessionId};**  **DatabaseHandler handler = DatabaseHandler.getInstance();**  **if (handler.isConnected()) {**  **// Check if session id with account id exists in Session id table**  **List<List<Object>> result = handler.get("accountId FROM Application.SessionId WHERE accountId = ? AND sessionId = ?", info, 1);**  **if (result.size() >= 1) {**  **return new Packet(PacketResult.SUCCESS);**  **} else {**  **return new Packet(PacketResult.BAD\_REQUEST);**  **}**  **}**  **return new Packet(PacketResult.CONNECTION\_ERROR);**  **}** |

The following function is interesting as it handles the deletion of an invoice.

Looping through every one of the invoice items before deleting the invoice. This is to preserve the quantity count of the product

|  |
| --- |
| **/\*\***  **\* Deletes an invoice with the given invoice id**  **\* @param invoiceId**  **\* @return**  **\*/**  **public static Packet deleteInvoice(int invoiceId) {**  **Packet companyVerify = Account.companyVerify();**  **if (companyVerify.getResult() == PacketResult.SUCCESS) {**  **DatabaseHandler handler = DatabaseHandler.getInstance();**  **if (handler.isConnected()) {**  **Object[] args = {invoiceId};**  **// Check if there is any invoice items**  **List<List<Object>> invoiceItemList = handler.get("productId FROM Application.InvoiceItem WHERE invoiceId=?", args, 1);**  **if (invoiceItemList.size() == 1) {**  **List<Object> invoiceItems = invoiceItemList.get(0);**  **for (int i = 0; i < invoiceItems.size(); i++) { // Loop through invoice items and remove them**  **Company.deleteInvoiceItem((int)invoiceItems.get(i), invoiceId);**  **}**  **}**  **boolean success = handler.delete("Invoice", "invoiceId = ?;", args);**  **if (success) {**  **return new Packet(PacketResult.SUCCESS);**  **}**  **return new Packet(PacketResult.DATABASE\_ERROR);**  **}**  **return new Packet(PacketResult.CONNECTION\_ERROR);**  **}**  **return companyVerify;**  **}** |

Lastly, the following function is interesting as it handles the adding of items into an invoice and subtracting from the quantity of the product.

This function really highlights the use of the class Packet to provide feedback to the user.

|  |
| --- |
| **/\*\***  **\* Adds invoice item to invoice by checking the quantity requested and the**  **\* quantity in stock After that, reduce product quantity and add invoice**  **\* item to invoice. If another invoice item exists inside of the invoice**  **\* then increment it**  **\***  **\* @param quantityRequested**  **\* @param product**  **\* @param invoice**  **\* @return**  **\*/**  **public static Packet addInvoiceItem(int quantityRequested, Product product, int invoiceId) {**  **DatabaseHandler handler = DatabaseHandler.getInstance();**  **if (handler.isConnected()) {**  **Packet quantityPacket = product.getQuantity(companyId);**  **if (quantityPacket.getResult() != PacketResult.SUCCESS) {**  **return quantityPacket;**  **}**  **int productQuantity = (int) quantityPacket.getInformation();**  **if (productQuantity >= quantityRequested) {**  **if (handler.isConnected()) {**  **// Reduce**  **Object[] args\_1 = {productQuantity - quantityRequested, product.getProductId()};**  **handler.update("Application.Product SET quantity=? WHERE productId=?", args\_1); // Update quantity**    **Object[] args\_2 = {invoiceId, product.getProductId()};**  **// Check if already exists inside of invoice**  **List<List<Object>> qInformation = handler.get("invoiceItemId, quantity FROM Application.InvoiceItem WHERE invoiceId=? AND productId=?", args\_2, 1);**  **if (qInformation.size() == 1) {**  **List<Object> information = qInformation.get(0);**  **int invoiceItemId = (int) information.get(0);**  **int invoiceItemQuantity = (int) information.get(1);**  **Object[] args\_3 = {invoiceItemQuantity + quantityRequested, invoiceItemId};**  **boolean success = handler.update("Application.InvoiceItem SET quantity=? WHERE invoiceItemId=?", args\_3);**  **if (success) { // Add item**  **return new Packet(PacketResult.SUCCESS);**  **}**  **return new Packet(PacketResult.DATABASE\_ERROR);**  **} else {**  **Object[] args\_4 = {0, invoiceId, product.getProductId(), quantityRequested};**  **boolean success = handler.insert("InvoiceItem(invoiceItemId, invoiceId, productId, quantity) VALUES (?,?,?,?)", args\_4);**  **if (success) { // Add item**  **return new Packet(PacketResult.SUCCESS);**  **}**  **return new Packet(PacketResult.DATABASE\_ERROR);**  **}**  **}**  **return new Packet(PacketResult.CONNECTION\_ERROR);**  **}**  **return new Packet(PacketResult.BAD\_REQUEST);**  **}**  **return new Packet(PacketResult.DATABASE\_ERROR);**  **}** |

## Tests:

**Test Case – Sign up Company**

|  |  |
| --- | --- |
| Name | TC-01 |
| Requirements | Secure |
| Precondition | User must be connected to database |
| Steps | User enters login screen  User enters the username and password  User submits the information  System brings user to company menu |
| Expected Results | User has logged in and is in the company menu |
| Actual Results | User has logged in and is in the company menu |

**Test Case – Login Customer**

|  |  |
| --- | --- |
| Name | TC-02 |
| Requirements | Secure |
| Precondition | User must be connected to database |
| Steps | User enters login screen  User enters the username and password provided by the company  User submits the information  System brings user to customer menu |
| Expected Results | User has logged in and is in the customer menu |
| Actual Results | User has logged in and where the company menu is supposed to be |

**Test Case – Add Invoice**

|  |  |
| --- | --- |
| Name | TC-03 |
| Requirements | Fast, Correct |
| Precondition | User must be connected to database, User must be logged into a company account |
| Steps | User clicks “Create new Invoice” button  User selects a customer from the customer combo box  User enters in a date  User submits the information by pressing “Ok”  System updates and shows newly created invoice |
| Expected Results | Newly created invoice inside of invoice table |
| Actual Results | Newly created invoice inside of invoice table |

**Test Case – Amend Invoice**

|  |  |
| --- | --- |
| Name | TC-04 |
| Requirements | Fast, Correct |
| Precondition | User must be connected to database, User must be logged into a company account, User must have selected an invoice |
| Steps | User clicks “Amend” button  User changes the customer in the customer combo box  User changes the date box  User submits the information by pressing “Ok”  System updates and shows invoice information changed in DB tables |
| Expected Results | Shows invoice information changed in DB tables |
| Actual Results | Shows invoice information changed in DB tables |

**Test Case – Edit Product Information**

|  |  |
| --- | --- |
| Name | TC-05 |
| Requirements | Fast |
| Precondition | User must be connected to database, User must be logged into a company account |
| Steps | User clicks on a slot for the product (eg. Click on a slot at the column called quantity)  User changes the quantity amount.  User unselects the quantity slot.  System updates and shows the new value |
| Expected Results | Shows product value changed in DB tables |
| Actual Results | Shows product value changed in DB tables |

**Test Case – Add Invoice Item**

|  |  |
| --- | --- |
| Name | TC-06 |
| Requirements | Fast |
| Precondition | User must be connected to database, User must be logged into a company account, User must have selected an invoice, An invoice item must have already been added into the invoice |
| Steps | User clicks “Amend” button  User on “Add new” button  User chooses same product from product combo box and puts a quantity of 5  User submits the form.  System updates and invoice item’s quantity increased by 5 |
| Expected Results | Invoice item’s quantity increased by 5 |
| Actual Results | Invoice item’s quantity increased by 5 |