

---

## SCHOOL OF SCIENCE AND TECHNOLOGY

FINAL ASSESSMENT FOR THE BSC (HONS) INFORMATION TECHNOLOGY; BSC (HONS) COMPUTER SCIENCE; YEAR 2

ACADEMIC SESSION 2021; SEMESTER 3

PRG2104: OBJECT ORIENTED PROGRAMMING

Project

DEADLINE: Week 14

---

### INSTRUCTIONS TO CANDIDATES

- This assignment will contribute 50% to your final grade.
- This is an individual assignment.

#### **IMPORTANT**

The University requires students to adhere to submission deadlines for any form of assessment. Penalties are applied in relation to unauthorized late submission of work.

- Coursework submitted after the deadline but within 1 week will be accepted for a maximum mark of 40%.
- Work handed in following the extension of 1 week after the original deadline will be regarded as a non-submission and marked zero.

**Lecturer's Remark** (Use additional sheet if required)

Saurabh Varughese M. Kovoor  
I..... (Name) 20017604 std. ID received the assignment and read the  
comments..... SVMK 16/7/2021 (Signature/date)

#### **Academic Honesty Acknowledgement**

Saurabh Varughese M. Kovoor  
"I ..... (student name). verify that this paper contains entirely my own work. I have not consulted with any outside person or materials other than what was specified (an interviewee, for example) in the assignment or the syllabus requirements. Further, I have not copied or inadvertently copied ideas, sentences, or paragraphs from another student. I realize the penalties (*refer to page 16, 5.5, Appendix 2, page 44 of the student handbook diploma and undergraduate programme*) for any kind of copying or collaboration on any assignment."

SVMK 16/7/2021  
..... (Student's signature / Date)

## Overview

The aim of this project is to achieve the learning outcomes of [d] of this subject as mentioned in the syllabus, your role is to analyse, apply, and design a software application using object oriented programming. You also need to demonstrate your work at the time of submission. This **overall** assignment **mark** will contribute **50** % of your final grade.

(You must fill the Assignment Reflection form and submit to Lecturer during presentation.)

### ASSIGNMENT SPECIFICATION

<b>Learning Outcome Being Assessed</b>	1. Write Computer programs that use: object-oriented programming concepts - inheritance, polymorphism, GUI programming - event-driven programming, layout managers.
<b>Submission Deadline</b>  <b>Microsoft Team Submission</b>	<b>Monday, (Week 14) by 4.00p.m.</b>  Late submission will be capped to 40% (unless a concrete reason is provided).  Create a submission folder named "Project_yourID". Put your project folder and documentation report into this submission folder. Zip it and submit this zipped file into the Microsoft Team.
<b>Outline of Problem</b>	This assignment <b>stipulates</b> the design of a system by the identification of the classes required and the relationships among them.  Students are required to demonstrate the ability to apply their knowledge on inheritance and polymorphism in their implementation of the solution. Student also need to make use of scalaFX GUI library to create an GUI Application.
<b>Detail Question</b>	Propose a standalone system such as the following 1. Calculator 2. Scientific Calculator 3. Library Management System 4. Todo List 5. Any Personal Games 6. ...  The propose system should have least four use cases. For examples: A Library Management System should have at least four following use case: <ul style="list-style-type: none"><li>• Check In</li><li>• Check Out</li><li>• Search Book</li><li>• View Book</li></ul> This system should utilize object oriented programming concept in designing and developing.

	<p>You should proposed additional classes or trait to achieve elegance design.</p> <p>You are required to design your own user interface for ease of use.</p> <p>If you refer to any sources from youtube, github or other code repository in creating the application, please do citation. Any work that is not created by you will not be consider.</p>
<b>What you should hand in</b>	<p>The following items are to be handed in:</p> <ul style="list-style-type: none"> <li>• A cover page (use the template provided).</li> <li>• A documentation <b>report</b> includes the UML diagrams that describe the classes/objects identified from problem domain, and their relationships among these classes/objects used in the program. – in <b>hardcopy</b></li> <li>• A description that show the program is working for all the 4 features.</li> <li>• An A4 page to be written by <b>you on the</b> <b>personal reflection</b> that includes: <ul style="list-style-type: none"> <li>○ A description of how you applied the above object oriented concepts in your assignment.</li> <li>○ The problems encountered during this assignment and how you solved these problems.</li> <li>○ An evaluation of the strengths and weaknesses of your submitted work.</li> <li>○</li> </ul> </li> <li>• The project/solution files, including the source code for the Item class, all pre-compiled classes, test driver program and application program. – in <b>softcopy</b></li> </ul> <p>NOTE: Submitting the assessment means you have agreed that your work is original and comply with the rules and regulations (refer to Academic Impropriety)</p>
<b>Paper Size / Format</b>	<p>Paper size                      A4 (Use only one side of the paper)</p> <p>For the personal reflection write-up,</p> <p><b>Paragraph format</b>                      <b>1.5-line spacing</b></p> <p><b>Font size</b>                                      <b>12 points</b></p>

<b>Academic Impropriety</b>	<p>Sunway University takes a strong stand on plagiarism. Any students found to have copied work, colluded or presented work that is not their own will be punished under the terms stated in the rules and regulations booklet. Students are permitted to use 3rd party components, however all such code must be well described and credit awarded to the respective owner. Students must also ensure that the majority of source code is their own, and that the core algorithms are their own work. The use of copyright materials is forbidden.</p> <p>*subject to change anytime without prior notification</p> <p><b>The work that you submit must conform to those regulations.</b></p>
<b>Assessment: Report</b>	<p>Contributes 50% to the overall final assessment mark.</p> <p>Refer to ASSESSMENT CRITERIA FOR Project table for further elaboration of marking distribution.</p>

## ASSESSMENT CRITERIA FOR Project

Mark / General Impressi on	Area / Assessment Criteria							
	Class Definition and Design			Application Program		Style	Use of Third-Party Library	Documentation
	Fulfillment of Requirements  (x2)	Relationships among classes	UML Class Diagram	Fulfillment of requirements  (x2)	GUI Implementation	Naming Convention	Mastery	Documentation (Report)
5  Excellent	(a) Correct and complete <ul style="list-style-type: none"> <li>Classes – both basic classes and the “collection” class</li> <li>Identification of data fields, visibility modifiers and types.</li> <li>Constructors</li> </ul> (b) Additional features / operations provided. (c) Originality and Uniqueness	Correct application of all the following concepts: <ul style="list-style-type: none"> <li>inheritance</li> <li>polymorphism</li> <li>abstract class</li> <li>generic programming.</li> </ul>	Perfectly correct diagram. <ul style="list-style-type: none"> <li>All notations are correctly used.</li> <li>Class members are complete, and</li> <li>The diagram is consistent with the class design.</li> </ul>	The following are provided <ul style="list-style-type: none"> <li>Menu navigation</li> <li>Execution of all the required operations</li> <li>Input validations</li> <li>Completely correct, efficient and elegant use of programming constructs.</li> <li>Methods are extensively used to achieve complete modular programming.</li> </ul>	Illustrated an excellent mastery in Event Driven Programming and correct use of layout classes.  GUI Components and layout component are design and use correctly.	Full adherence to naming convention with appropriate, meaningful and correctly spelt identifier names.	An excellent correct use of third-party libraries to solve problem.  No errors in utilizing the third-party libraries.  Demonstrated in depth understanding of third-party library model.	<ul style="list-style-type: none"> <li>Complete and well written documentation.</li> <li>All Required section is included.</li> <li>Table of content is formatted properly.</li> </ul> Very few typo or spelling mistake.
4  Very Good	Correct and complete <ul style="list-style-type: none"> <li>Classes – both basic classes and the “collection” class</li> <li>Identification of data fields, visibility modifiers and types.</li> <li>Constructors</li> </ul>	Correct application of the following concepts: <ul style="list-style-type: none"> <li>inheritance</li> <li>polymorphism</li> <li>abstract class</li> </ul>	<ul style="list-style-type: none"> <li>Complete with only one very minor error in notations used.</li> <li>Diagram is consistent with class design.</li> </ul>	The following are provided <ul style="list-style-type: none"> <li>Menu navigation</li> <li>Execution of all the required operations</li> <li>Correct and efficient use of</li> </ul>	Illustrated a good mastery in Event Driven Programming and correct use of layout classes.  GUI Components and layout component are	Adherence to naming convention with meaningful, appropriate and correctly spelt identifier names.	Make a good use of some third-party libraries to solve problem.  No errors in utilizing the third-party libraries.	<ul style="list-style-type: none"> <li>Complete and well written with only one very minor error in documentation.</li> <li>Not all required section is included only miss one section.</li> </ul>

	<ul style="list-style-type: none"> <li>Quite original and uniqueness</li> </ul>			<p>programming constructs.</p> <ul style="list-style-type: none"> <li>Methods are used to achieve a high degree of modular programming.</li> </ul>	design and use correctly with minor error.		Demonstrated in good understanding of third-party library model.	<ul style="list-style-type: none"> <li>Table of content is formatted reasonably. Few typo or spelling mistake.</li> </ul>
3 Average	<p>Correct and complete</p> <ul style="list-style-type: none"> <li>Basic classes</li> <li>Identification of data fields, visibility modifiers and types.</li> <li>Constructors</li> <li>Partial originality and uniqueness</li> </ul>	<p>Correct application of the following concepts:</p> <ul style="list-style-type: none"> <li>inheritance</li> <li>polymorphism</li> </ul>	Quite complete with not more than two minor errors in notations used. Diagram is consistent with class design.	<p>The application program demonstrates the correct execution of all the required operations</p> <p>One or two minor errors.</p>	<p>Illustrated an average mastery in Event Driven Programming and correct use of layout classes.</p> <p>GUI Components and layout component are design and use correctly with some error.</p>	General adherence to the naming convention with one or two minor errors.	<p>Make use of some third-party libraries to solve problem.</p> <p>Minor errors in utilizing the third-party libraries.</p> <p>Demonstrated in average understanding of third-party library model.</p>	<ul style="list-style-type: none"> <li>Quite complete and good written with only few minor error in documentation.</li> <li>Not all required section is included only miss few section.</li> <li>Table of content is formatted. Few typo or spelling mistake.</li> </ul>
2 Poor	<p>Some errors or 1 incomplete basic class.</p> <p>No originality with some uniqueness</p>	Correct application of inheritance	Some errors in notation or diagram's consistency with class design.	<p>Incomplete application program.</p> <p>Some errors.</p>	<p>Illustrated a poor mastery in Event Driven Programming and correct use of layout classes.</p> <p>GUI Components and layout component are design and use correctly with few error.</p>	Limited adherence to naming convention.	<p>Make use of a third-party libraries to solve problem.</p> <p>Some errors in utilizing the third-party libraries.</p> <p>Demonstrated in poor understanding of third-party library model.</p>	<ul style="list-style-type: none"> <li>Incomplete in documentation with few major error.</li> <li>Not all required section is included with missing lot of section.</li> <li>Table of content is partially formatted. Lot of typo or spelling mistake.</li> </ul>
1	Very major errors or more than 1 incomplete basic class.	Illogical inheritance hierarchy	Major errors in notation or diagram's	Grossly incomplete application program.	Illustrated a very poor mastery in Event Driven Programming and	Serious lack of adherence to	Make use of a third-party libraries.	<ul style="list-style-type: none"> <li>Grossly incomplete in documentation with major error.</li> </ul>

Very Poor			consistency with class design.	Very major errors.	incorrect use of GUI components.	the naming convention.	Major errors in utilizing the third-party libraries.	<ul style="list-style-type: none"> <li>• Major required section is not included.</li> <li>• Table of content is not formatted.</li> </ul> Lot of typo or spelling mistake.
-----------	--	--	--------------------------------	--------------------	----------------------------------	------------------------	--	--

## Table of Contents

<b>Introduction .....</b>	<b>2</b>
<b>Objectives .....</b>	<b>3</b>
<b>Requirements (SRS) or System Functionalities.....</b>	<b>3</b>
<b>UML Class Diagram .....</b>	<b>4</b>
<b>Results or Implementation of 4 Use Cases.....</b>	<b>5</b>
<b>Demonstration .....</b>	<b>8</b>
<b>Personal Reflection .....</b>	<b>9</b>
<b>Implementation of OOP Concepts.....</b>	<b>9</b>
<b>Problems During the Project.....</b>	<b>10</b>
<b>Strength and Weaknesses of the System .....</b>	<b>11</b>



## Introduction

For this Object Oriented Programming (PRG2104) final project, I aimed to create an inventory management platform for a fictional tech or computer store (TuringTech) using scalaFX.

Overall, the system has 8 main pages that the user can access, namely:

1. Login Dialog Page - the first page shown to the user, where the user logs in to the system
2. Registration Dialog Page - accessible through the login dialog page, where users can register and create an account with the system.
3. Home page/Dashboard page - the page shown after logging in.
4. Product page - the page that stores information about products, allowing users to view, add, update and delete product entries.
5. Product category page - the page that stores information about categories of products, allowing users to view, add, update and delete product category entries.
6. Supplier page - the page that stores information about suppliers, allowing users to view, add, update and delete supplier entries.
7. Order page - the page that stores information about orders, allowing users to view, add, update and delete order entries.
8. User page - the page that stores information about system users, allowing users to view, add, update and delete system user entries.

Basically, this is a CRUD application, where it performs the general functions of an inventory management system using the sql commands Create, Update and Delete. Thus, the use cases or functionalities I would like to highlight and test through this system are adding, updating and deleting as well, in this case to the database to store information about the TuringTech shop and its system, plus logging in and registering. Thus, the system utilises a database component known as scalikejdbc to handle storing and accessing the system data. You can find a more detailed explanation on the results and implementation of these use cases in the following section.

Overall, despite being branded for a tech shop, the design of the system allows for it to be used for just about any form of inventory management for any store that has and would like to track their products, product categories, suppliers, orders, and users.

## **Objectives**

As mentioned previously, the primary objective of this project is to create a simple and user-friendly inventory management system for a particular tech shop, in this case the TuringTech company. Thus, it aims to make storing, accessing, and updating various information about the shop simpler for the user.

Therefore, firstly, there's a crucial need to understand the common types of variables needed to be stored in each class or page of the system. Thus, this will require some initial research and brainstorming on the information the system should store and display. From there, we can determine the variables and the data types that can be used.

## **Requirements (SRS) or System Functionalities**

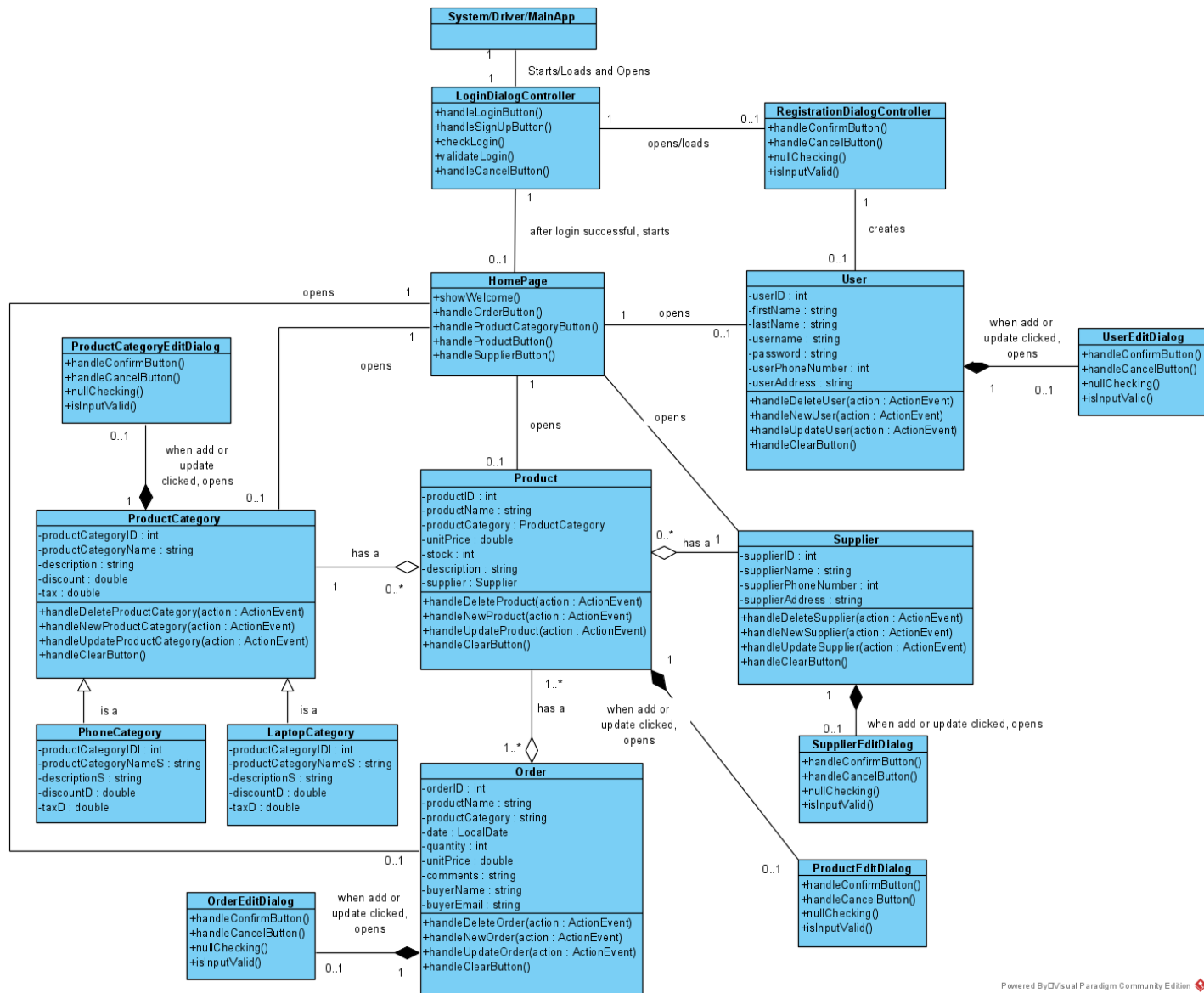
The system shall only be accessible by authorised users of the shop and system. Thus, for this, the system shall have a registration functionality, where users can enter necessary system user information (name, username, password, phone, email), to create a user account with the corresponding login credentials. This information will then be stored in the user class and page. After that, the system shall have a login system where the user can enter these login credentials.

The system shall allow the users to access the subsequent pages either through the homepage/dashboard, the main sidebar menu, or even through a combination of hotkeys (for accessibility).

On these pages, the system shall allow users to add, update or delete items at the corresponding tables. Then, the changes shall be made accordingly, and to the database using the scalikejdbc.

# UML Class Diagram

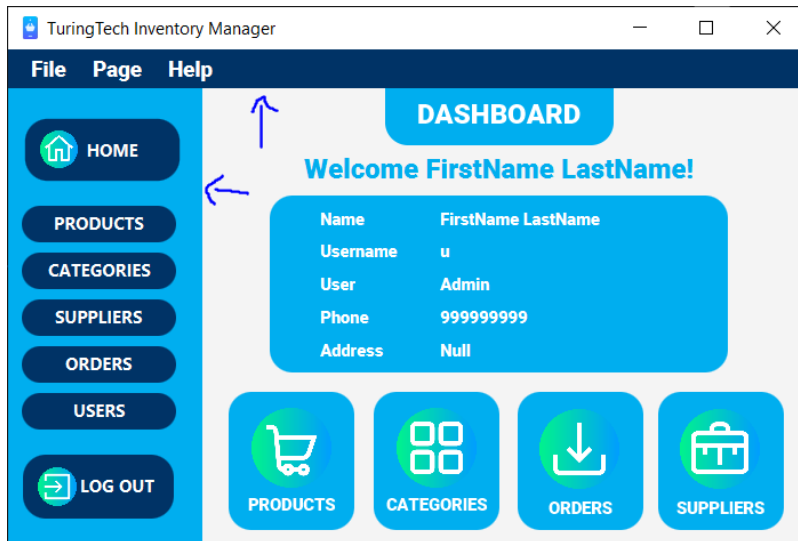
The following is a class diagram, that I created using visual paradigm, representing the relationship among the classes of the inventory management application. You may have to zoom in for a clearer view of the classes and the data. I've also attached the VPP file of the class diagram which you can open using the Visual Paradigm app, in case you need further checking.



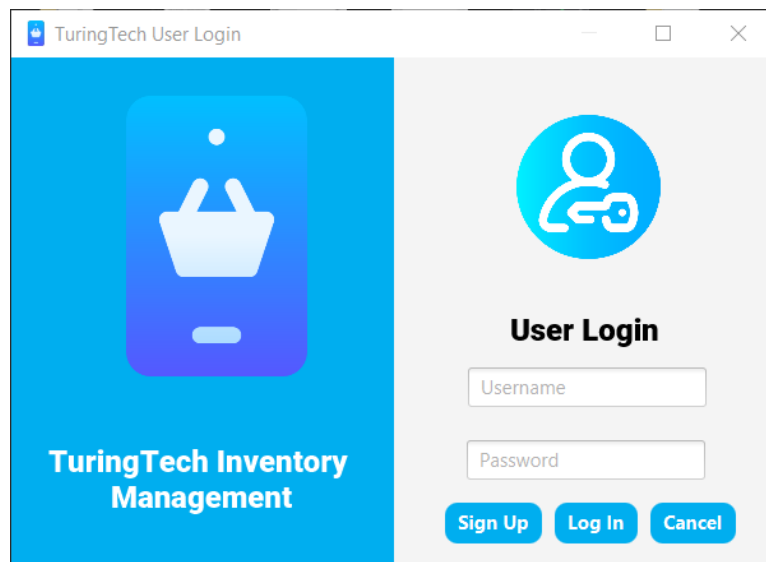
## Results or Implementation of 4 Use Cases

### 1. User Log In and Registration

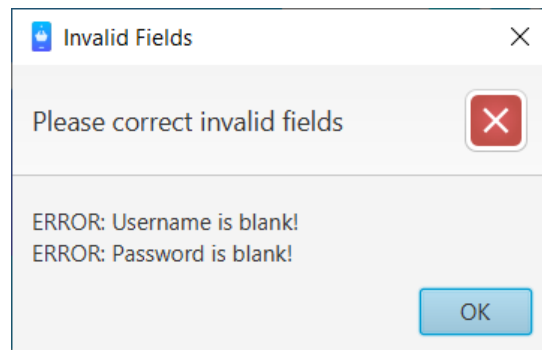
The system begins with displaying the Log In dialog. This log in page is displayed as a dialog so that it doesn't possess the sidebar menu and top navigation menu that's with the main rootMenu page. This allows for these menu to be appear consistently on every other page, and so that the functions will not be repeated in every page's controller class.



So, at this login page users can either choose to open the registration page, log in, or cancel and close the entire application.



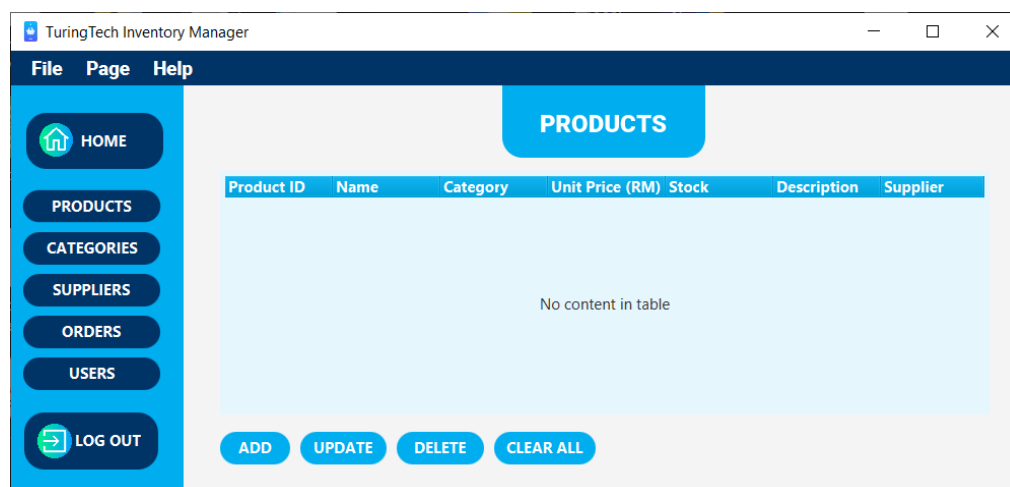
When entering the login credentials, first, input validation is performed to ensure that the user doesn't enter an invalid or blank input. If so, a corresponding error message will be shown, signifying whether the username field or password field has been left empty.



Then, the credentials are compared with the entries in the user table, if there exists a same combination of username and password. If so, the system successfully closes this login dialog and loads and opens the homepage as the user is permitted to enter. If the login credentials are invalid, a corresponding error message is shown.

## 2. Adding Items to TableView

This add function is available in all the pages except homepage, and functions similarly. At the page (e.g. Product page) the users can click the Add button to bring up Add/Update Dialog page.

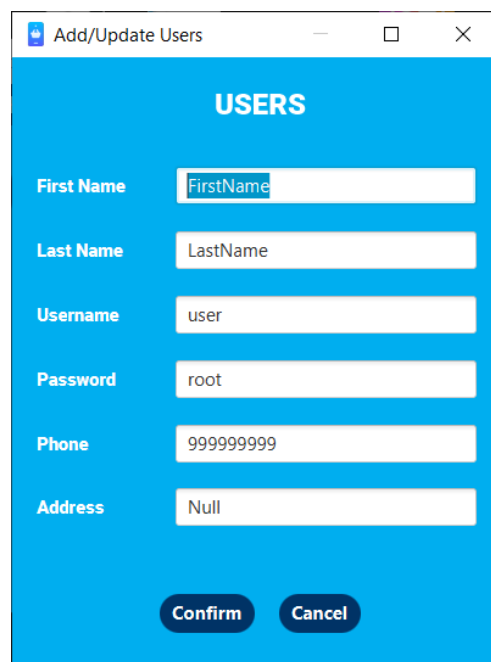


Here, users can enter the corresponding information, then data validation is performed to ensure the data fields are not empty and valid.

After that, the entries are saved to the database as a new record through the save function and added to the table.

### 3. Updating Items in a TableView

Similar to the add function, the Update button is available on most pages where data is displayed. Thus, the update functionality works similarly on these pages. At the page, when users select an entry from the tableview, they can click the Update button to bring up the Add/Update Dialog page.



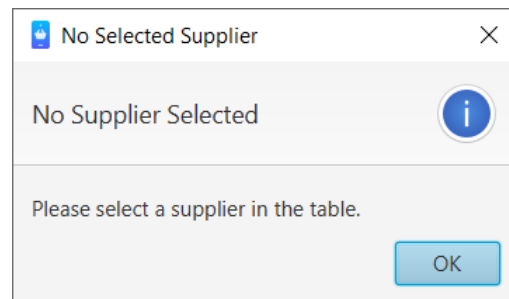
The screenshot shows a mobile application dialog titled "Add/Update Users". The dialog has a blue background and a white header with the title "USERS". Below the header, there are six input fields with labels: "First Name", "Last Name", "Username", "Password", "Phone", and "Address". The "First Name" field contains the text "FirstName", "Last Name" contains "LastName", "Username" contains "user", "Password" contains "root", "Phone" contains "999999999", and "Address" contains "Null". At the bottom of the dialog, there are two buttons: "Confirm" and "Cancel".

Here, the information from that record is available on the dialog page, and the user can make their changes. Once done, they can click the Confirm button, then the changes will be implemented to the database at the corresponding record in the table through the save function, and the changes will be applied to the table as well.

### 4. Deleting Items from a TableView

Similar to the update functionality, the delete button allows users to select an entry in the tableview and remove or delete it by clicking the button. After that, the specific record is deleted from the database and removed from the tableview.

If no records are selected and the user clicks this button, the system will display an appropriate error message.



Similarly, there's another functionality called Clear All, which completely clears the data in the table by simply clicking the Clear All button. This is to provide an easy way or one-click option for the system user or admin to completely remove all the data in the table.

After clicking it, the system will also display a confirmation message, to express that this action is irreversible and this is to protect the data from accidental erasures or clicks of the clear all button. Aside from clearing the data shown in the table, this function calls the deleteAll function in the corresponding model class which uses the sql command "truncate" to completely reset and clear the table in the database. Thus the change will be applied to the database as well, so in the next session or when the user opens the application again the table will be cleared.

## Demonstration

The following is a link to a demonstration video of the application which is hosted on YouTube. Through this video you can see the application being put to use and how its use cases function:

<https://youtu.be/vM1EnwZxsm8>

## **Personal Reflection**

### **Implementation of OOP Concepts**

#### **Abstract Class and Inheritance**

An example of the implementation of abstract class and inheritance in the application is the `productCategory` model class, which is abstract. Therefore, we can't instantiate a `productCategory` object but we can instantiate its classes that inherit from it. From there, I created 2 child classes called `phoneCategory` and `laptopCategory` model class which inherit from it. Thus, they have the data fields of the parent (`productCategory`) and have inherited the functions, especially the functions querying the `productCategory` table. This allows for the instantiations or objects of the `phoneCategory` and `laptopCategory` to be stored into the database. For this, at the `productCategory` page, there are 2 Add functionalities to add either a phone or laptop category to the table. These 2 child classes have unique data fields which are the tax and discount value. So, every phone product category that's created has a discount value of 20(%) and tax value of 6.5(%), whereas laptop product category has a discount value of 14(%) and tax value of 5.5(%). These 2 values cannot be updated through the update functionality, making it unique and consistent to objects of that particular product category.

#### **Polymorphism**

An example of polymorphism in this application is Static Polymorphism, which is also known as method overloading. This means that using the same function declaration has the same name but accepts different parameters of different data types. I used this in my program where I declared 2 functions in the `User.scala` class with the function name `getUserWithUserData()`, one that accepts string argument values which are to be the username, and another that accepts int argument values which are to be the userID. The first function is used to obtain the user record based on the entered username at the login page, in order to show the welcome message and user information on the homepage. The second one although isn't used in practice in the program, this provides an additional functionality or freedom in case we would like to call the same function and find the user record corresponding to the particular userID.

#### **Generic Programming**



One instance where I use generic programming is in the functionality of displaying the welcome message and user info at the system's homepage. For this we need to retrieve the particular user's (who just logged in) data, and display it in the homepage. Therefore, I created a function, `getUserWithUsername()` in the `User.scala` class which queries the user table for the User record using the username that they had entered in the login page. So, the return type of the function is an `Option[User]` type. So, the return value can be `Some` where it returns User values that are present in the user table or `None` if the table doesn't contain any user records for with that particular username.

Additionally, throughout the program you may notice instances of generic programming. One such main application of this is in assignment and instantiating variables/values where for reference types, I prefer using the `ObjectProperty` datatype with the different `Type` variables depending on the certain data. For example, for double values like in the product table for a product's unit price, I assign an `ObjectProperty` data type with `Double` type (`ObjectProperty[Double]`). This is because when using `DoubleProperty` we may get a mismatch error because conversion from `scalafx.beans.property.DoubleProperty` to `scalafx.beans.values.ObservableValue[Double, Double]` tends to be missing. This conversion is available for `String` and generic types, but may cause problems with primitive types like `Int`, `Long` and `Double`. Hence, as a workaround I use `ObjectProperty[Double]` instead.

## **Problems During the Project**

### **Ensuring the same menu options were available for every page except for the starting login page**

The application begins with the login page. However, the login page can't have the root menu, which consists of the top menu bar and side menu, as then a user can access the pages without logging in. This isn't good for the application's security.

Therefore, the solution to this was the login page was implemented as a dialog where the main root menu acts as the owner or root. By doing this, we can load and display the login page separately without displaying the root elements on it.

## **Closing the Login Dialog Doesn't Close the Entire Application**

Continuing from the previous problem, as the login page is a dialog on the root menu, this brings another problem when closing the login dialog. When closing it, this will bring up the root menu page with the top menu bar and side menu. This is also not good for the application security as a user can access the system pages without having to log in.

Therefore, to overcome this at the MainApp.scala class, at the showLoginDialog() function, using the onCloseRequest method, i can ensure that whenever a close request is performed on the login dialog, I can close the entire application using the Platform.exit() function. Therefore, the root menu will no longer be opened as the entire application is closed.

## **Strength and Weaknesses of the System**

### **Strength**

#### **1. Easy to use**

This is a non-functional requirement that I had initially planned for the application to achieve. I managed to achieve this through the use of 2 clear and indicative menus, one top menu and a sidebar menu. This makes it easy to navigate through the app as users can access every page of the system through these 2 menus. Plus, this menu design appears consistent on every page.

Along with that, the top menu provides additional accessibility as well, as the menu buttons are also mapped with corresponding easy-to-learn hotkeys or accelerators. Another way the system is easy to use is because there are well defined-error message and corresponding solutions. For example when adding an entry in a table, if an invalid data is inputted to the form, the form will or EditDialog will show where the corresponding error has been made, so the user knows where and how to rectify the error.

#### **2. Well-designed**

Additionally, the system possesses a clean and consistent design. For this, I prioritised using a monochromatic color scheme of light (#00aeef) and dark blue (#003366). This color scheme is

found to be more readable and easier to view on the eyes. Also I've added images to certain buttons for it to be more visually appealing and indicative of its function. This is achieved through the fxml property ImageView.

## **Weaknesses**

### **The pages can't access information from other pages**

The system could use a more sophisticated object relational mapping (ORM). Thus, allowing for easier access of information between different tables. For example, between the order table and the product table, a user can create an order and select and show a product's information by selecting its product ID. In future, this sort of ORM can be achievable through the application of Java persistence and Hibernate. However, this application is designed to record data for its products, product categories, users, suppliers and orders, and the current application still manages to achieve this. Although, with an ORM, we can make table entries more efficient and at the same time ensure that data consistency is preserved. Nonetheless, the project accomplishes its initial goal and requirements which is to to be able to record data about the TuringTech shop.