

60-311 Final Project Report

Team Potato Bakers

University of Windsor

Computer Science Department

Contents

List Of Figures.....	3
1.Introduction.....	5
1.1 Problem Statement	5
1.2 Project Overview.....	6
1.3 Release Notes Summarizing the Last Iteration	6
2.Software Requirements.....	7
2.1 Functional Requirements.....	7
2.2 Iteration 1 Requirements:.....	7
2.3 Iteration 2 Requirements:.....	8
2.4 Iteration 3 requirements:.....	8
2.5 Non-functional Requirements.....	9
2.6 Product requirements:.....	10
2.7 External requirements:.....	10
2.8 Use Case Diagram.....	10
3. Design specification.....	11
3.1 Software Architecture.....	11
3.2 Sequence Diagrams and Related use cases.....	12
3.2.1 Use case 1: Print Bill.....	12
3.2.2 Use case 2: Add item to order.....	14
3.2.3 Use Case 3: Login to the system.....	16
3.2.4 Class diagram.....	18
4. Technical documentation.....	20
4.1 Programming language and platform.....	20
4.2 Database Table and Schema.....	20
4.2.1 Admins:.....	21
4.2.2 Orders:.....	21
4.2.3 Category:.....	22
4.2.4 FlamedN (and others):.....	23
5. User Documentation.....	24

5.1 Add Item to Order.....	24
5.2 Remove Item From Order.....	26
5.3 Print bill - Print Receipt.....	27
5.4 Make Payment	28
5.5 Split Bill	29
5.6 Add Tip.....	29
5.7 Add Discount	30
5.8 Login - Logout.....	31
5.9 Admin Interface (get range of orders).....	32
5.10 New Order.....	32
6. Software Testing.....	33
6.1 Overview of testing.....	33
6.2 Regression testing.....	33
6.3 Unit testing.....	33
6.4 Software inspection and validation.....	34
6.5 Release testing.....	34
Summary of total work done.....	35
8. References.....	38

List Of Figures

Figure 1: Use case diagram.....	9
Figure 2. System Architecture.....	10
Figure 3: Print Bill Sequence diagram.....	13
Figure 4: Add Item Sequence diagram.....	15
Figure 5: Login sequence diagram.....	19
Figure 6: Class Diagram.....	20
Figure 7: Employee database table.....	22
Figure 8: Order history table.....	23
Figure 9: Category table.....	24

Figure 10: FlamedN Menu Table.....	24
Figure 11: Adding items to order.....	25
Figure 12: Removing items from order.....	26
Figure 13: Making payment.....	27
Figure 14: Add tip.....	29
Figure 15: Add discount.....	30
Figure 16: Login.....	31
Figure 17: Using Admin Interface.....	32
Figure 18: Unit test using Visual Studio.....	34

1. Introduction

This project is a system designed to calculate and print the bill for a paying customer of a particular restaurant. The restaurant chosen was Sushi-One Express (Located: 337 Ouellette Avenue, Windsor, ON). See website: <http://www.sushi-one-express.com/>

This final report discusses in details about Potato Bakers' software development process of a bill calculating system for Professor Dan Wu. It outlines the problem statement given by the client. Project overview is a quick summary of the project. The final iteration details are given. Software requirements document is incorporated which includes consistent and unambiguous functional requirements and some software qualities. Some design qualities consist of correctness, efficiency, robustness, and user friendliness. Design specification document deliberates on software architecture description with diagrams, three sequence diagrams, class diagrams, and their relationships showing to all the important classes. Technical documentation includes what programming languages, platforms we used. It further shows algorithms, tools, and environment was used to create this project. Lists of the table for the database that we used to implement the project are also included. For quick learning user documentation was created as an easy-to-use guideline with helpful screenshots of the program for quick learning. Lastly, the report reviews the team's software test plan including the test approach, features tested, and the environment for the testing.

1.1 Problem Statement

Potato Bakers was assigned a task to create a program used by cashiers to calculate the bill for their clients. This is well-known scenario to most people. The customer goes into a restaurant, orders dishes they like, eats the food represented, at the end asks the server to bring the bill, upon arrival of the bill, customer quickly checks if everything is correct and pays the bill. If the server deserves a tip, a tip is added to the bill, either using cash or percentage. Sometimes a customer may bring in a coupon which stores use to promote their restaurants; if valid then the server deducts the total amount or percentage from the final bill for that customer.

Previous orders are stored in the database so the client can use it for as a reference for things such as income tax. Only the admin can login and look at those details.

1.2 Project Overview

The program the team created is dynamic, and can be used for any restaurant menu. For demo, the team used Sushi One restaurant located in downtown Windsor. C Sharp was implemented using Windows Visual Studio to meet the requirements of the above problem statement. The team used Microsoft's Visual Studio Online servers as well as Redmine to store source code and manage backlog. The database used was SqlServer for menu items and storing the bills after the customer was checked out.

1.3 Release Notes Summarizing the Last Iteration

New Features:

- User can now split their bill. The items they will split on their bills will be divided by half which will be considered as a discount.
- Discounts can be added to the bill by amount or percentage. For example if there is a deal of getting 5 dollars off when spending over 30 dollars or 5% off from your bill this can be done by using our discount buttons.
- New order button is added to our GUI which will clear everything off the screen and allow the workers to create a new order easily.
- We adjusted our program to print the bill for the customer. This does not include the payment and it won't be recorded in the database. After the payment is received two copies of receipts will be printed. Customer version for the customer to keep and merchant version for the restaurant to keep for records.

- Each employee will receive their own login and logout information. This will help keep track of each order and the tips received by employees. This feature is also required for security reasons.
- Fixed the admin interface by preventing the errors that occurred when checking the orders made previously. Now the manager can easily check all the previous orders made throughout the years.

2. Software Requirements

2.1 Functional Requirements

Functional requirements describe what the system shall do, user expectations of system functionalities, as well as describe how the system will act in certain scenarios. Functional requirements were gathered in an iterative way, usually collected during meetings with the client and later verified by the client. Each iteration, new requirements were added and implemented.

2.2 Iteration 1 Requirements:

The main objective for this iteration was to design and complete main functionalities of the system in a primitive prototype.

Main requirements:

1. Add item to order: The user must be able to select and add a particular menu item to the current customer's bill
2. Remove item from order: The user must be able to remove a particular item from the current customer's bill

3. Calculate totals: The system must be able to calculate the appropriate totals based on the price of items ordered and tax.
4. Print bill: The user must be able to print the current customer's bill, including a summary of all items ordered, the subtotal amount, tax amount and total amount due.

2.3 Iteration 2 Requirements:

The main objective for this iteration was to design and complete transaction based functionalities and payment methods

Main Requirements:

1. Store transaction to database: The system must record past transactions of customers and store them to a database as well as all information about their order including subtotal, tax, tip and total amount due
2. Tipping system: The user must be able to add customer's tips as part of an order
3. Payment methods: The system must be able to accept three methods of payment (Cash, credit, and debit) as well as be able to accept partial payments in cash and debit/credit.
4. Payment calculating system: The system must be able to calculate and update totals every time new information is added, including tips. The system must also be able to recognize when a payment has been made in full, and when a payment is not yet complete.

2.4 Iteration 3 requirements:

The main objective for this iteration was general redesign and bug fixing.

Main requirements:

1. Print bill to pdf file: The bill must be printed to a pdf file of appropriate size and format
2. Print receipt: The system must now separate the printing of bills and receipts. Bills are printed before the customer has paid in full, and receipts are printed after the customer has paid in full.

3. Print two copies of receipt (to pdf file): The system must be able to print two copies of the receipt once a payment has been made in full. One copy is to be retained for the restaurant, the other is given to the customer.
4. New Bill: The system must be able to reset the current order, or generate a new order, without having to close down the system and restart it.
5. Cash rounding according to law: When full payments are made in cash, the amount due must be rounded to the nearest nickel to accommodate the inability to pay with pennies.

In addition to these requirements, some functionality was suggested to the client to add usefulness and improve security. These requirements include:

1. Employee database: The system must be able to access the basic information about employees and their accounts, including full name, username, and password, and retrieve each of these from the database they are stored in
2. Employee login: The system must not be available for anyone to tamper with, employees must only be able to use the system after a successful login with valid employee information
3. Admin Interface: The system must allow the administrator to view information about sales in the past, including total number of sales, total amount of money made, and the average sale within a chosen time frame.
4. Admin interface security: The system must check to make sure that the employee logged in has admin privileges before allowing access to the information of past sales.

2.5 Non-functional Requirements

In order to assure software quality, it must adhere to non-functional requirements. These requirements are usually defined by standards and describe the limitations of the system outside of its functionalities.

2.6 Product requirements:

1. The system must only be available to valid employees of the restaurant
2. The system must be available during regular open hours of the restaurant
3. The system must be easy to use for employees
4. The system must be easy to teach to employees once deployed
5. The user groups must consist of both administrators and regular employees

2.7 External requirements:

1. The system must calculate tax according to provincial law
2. The system must accommodate for the elimination of the penny when accepting cash payments

Following these requirements gave the system additional security as well as relevance according to law in its geographical location.

2.8 Use Case Diagram

The following diagram illustrates the high level functionality of the software system

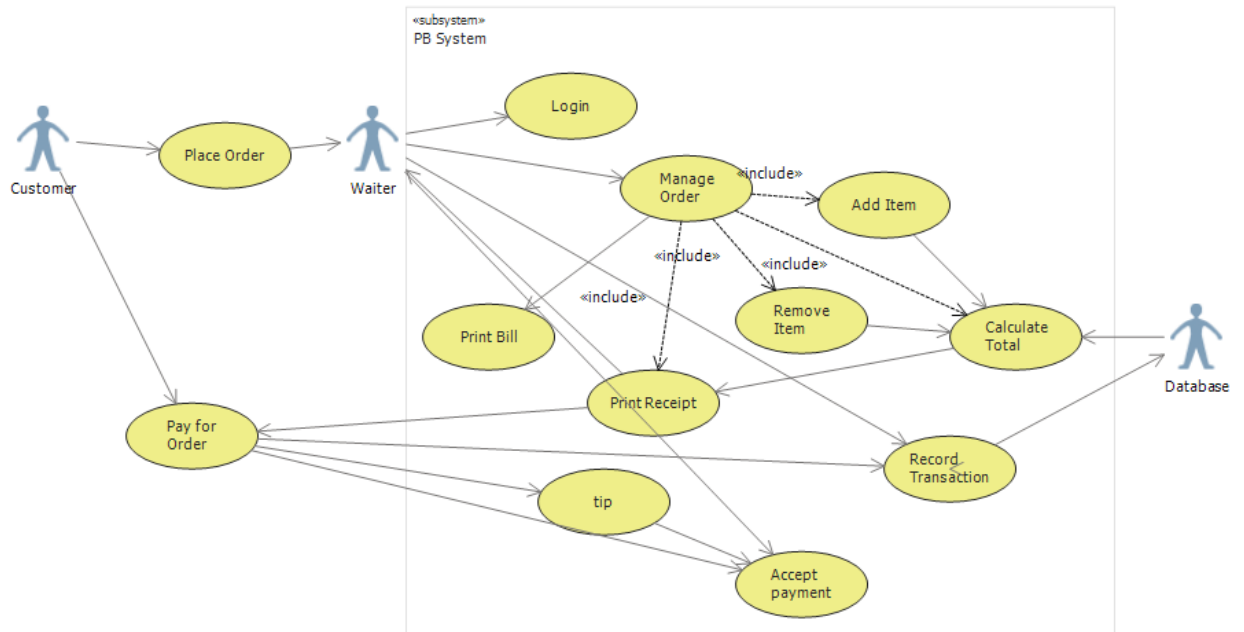


Figure 1: Use case diagram

3. Design specification

3.1 Software Architecture

System

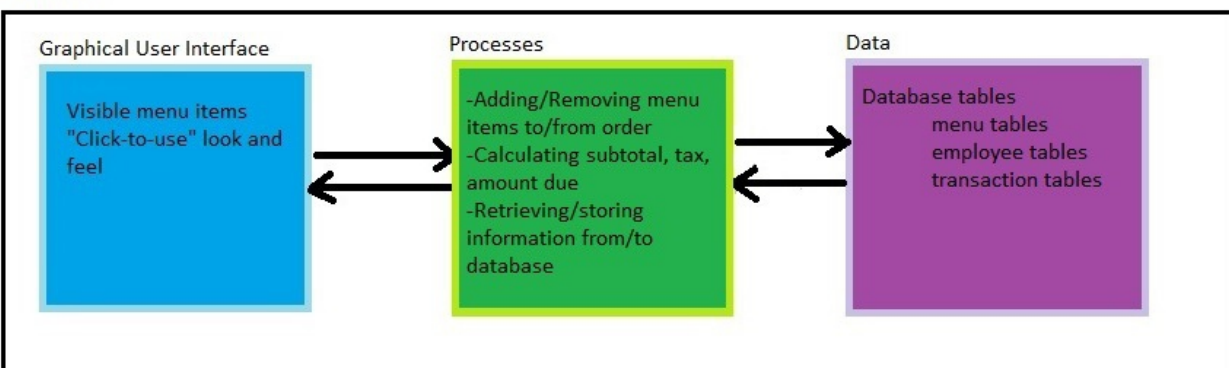


Figure 2. System Architecture

The software system takes a layered approach in its architectural design. Each tier or “layer” are arranged such that it only interacts with the “layers” directly above or below it. As pictured above, there are three main tiers or “layers” that are part of the system. These are:

1. The Visual Tier: Including the MainWindow, LoginDialog, AdminInterface classes as well as any visual aspect of the software. This part of the software is meant to interact directly with the user, and the classes of this portion can be seen as controllers, handling user input and allocating tasks to the classes of the next tier.
2. The Process Tier: All processes separate from input handling are taken care of in this tier. These processes include updating totals, refreshing data, filling and manipulating lists, and retrieving information from the data classes in the third tier.
3. The Data Tier: This tier handles all database-related needs. This tier includes classes that interact directly with the database in the back end, and pass the information to the process tier for handling.

3.2 Sequence Diagrams and Related use cases

Sequence diagrams were helpful in modelling the solution when designing the software system. Each use case could have a simple sequence diagram modelling how the programmer could approach the solution using lifelines as classes, and message labels as operations within those classes. Sequence diagrams also provided insight for class diagrams. Below is a sample of three use cases and their corresponding sequence diagrams.

3.2.1 Use case 1: Print Bill

Use Case		Print Bill
Description		This use case allows the user to print the bill to bring to the customer to inform the customer of the total amount he or she owes.
Actor		User
Preconditions		The user clicks the “Print Bill” button before the payment

		has been made.
Main Flow	A1	The use case begins when the user presses the button labeled "Print Bill"
	A2	The system creates a "Document" object to store the bill text in using the open source library iTextSharp
	A3	The system opens the document object
	A4	The system creates a paragraph object using the open source library iTextSharp
	A5	The system sets the paragraph object's text attribute by taking text from the order preview and totals list boxes and appending them to the paragraph
	A6	The system then adds the paragraph to the document object
	A7	The system closes the document file object and terminates the use case
Exceptions	A8	An error message is displayed if the document file could not be opened or written to properly.
Post conditions		None
Special Requirements		The bill can only be printed before a payment has been made

Description		This use case allows the user to add items to the customer's order
Actor		User
Preconditions		The user selects the desired item from the appropriate menu list and then clicks the "Enter" button after setting the desired quantity
Main Flow	A1	The user case begins when the user presses the "Enter" button
	A2	The system uses the input string to determine which item is to be added and how many
	A3	The System creates a new OrderItem and stores it in the order list of existing items for customer's order
	A4	The OrderItem consults the DataClass to get its own item info
	A5	The DataClass directly accesses the database to retrieve the information such as price and item name
	A6	The OrderItem stores the information
	A7	The system updates the totals for the order
Exceptions	A8	An Error message is displayed if the item doesn't exist or if the input string is not properly formatted
Special Requirements		The user must be logged in to add items to the order

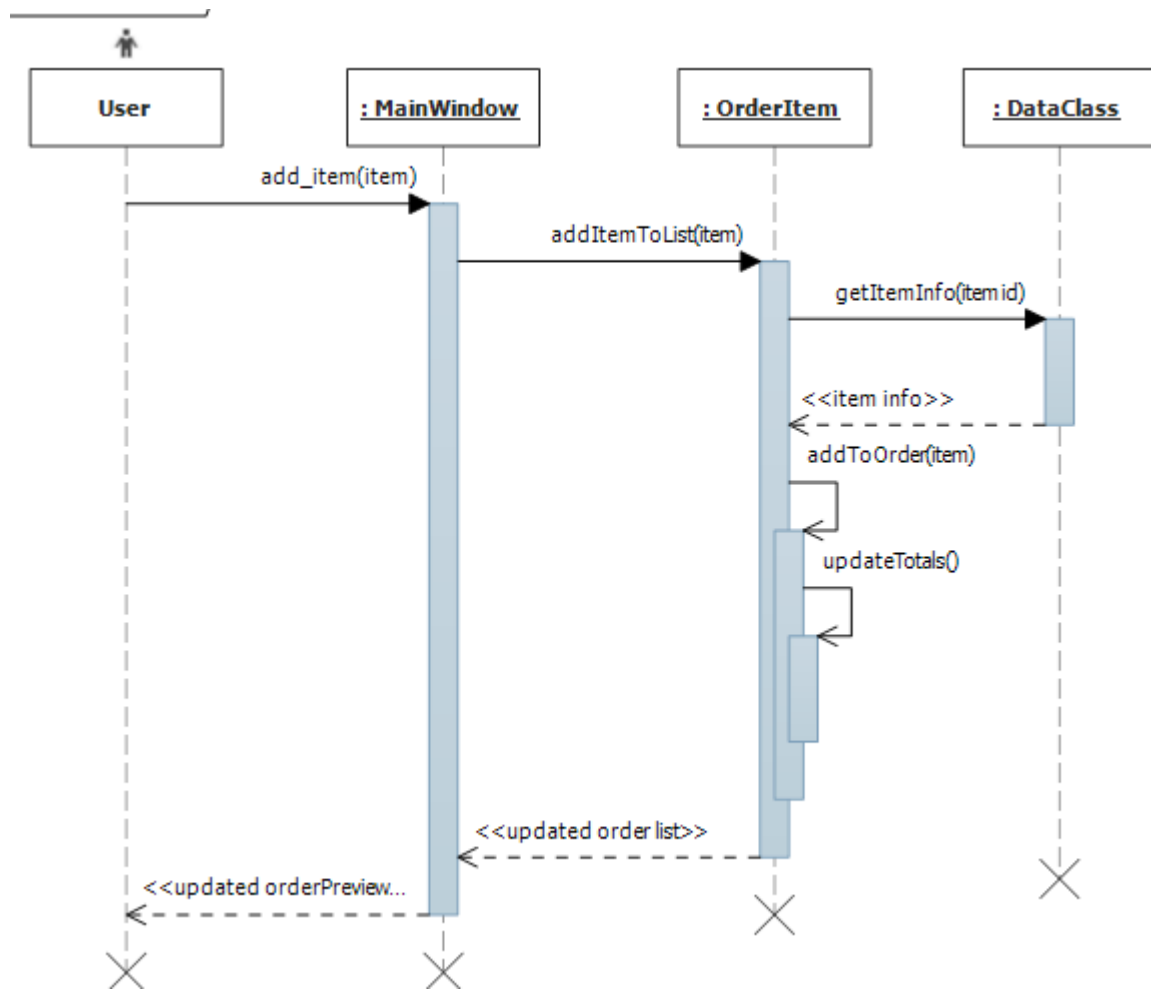


Figure 4: Add Item Sequence diagram

3.2.3 Use Case 3: Login to the system

Use Case		Login
Description		This use case allows the user to login to the system which in turn allows the system to be used
Actor		User
Preconditions		The user must click the "Login" button
Main Flow	A1	The use case begins when a dialog is popped up with textboxes for the user to input username and password
	A2	The user clicks "OK" after entering the appropriate

		information
	A3	The system consults the DataClass to see if the input username and password are a match in the system
	A4	If there is a match, the system displays a successful login message and loads the menus, allowing the system to be used. This terminates the use case
Alternate flow	A5	If the input username and password find no match in the database, then a login failed message will be displayed, and no menus will be loaded
Special Requirements		None

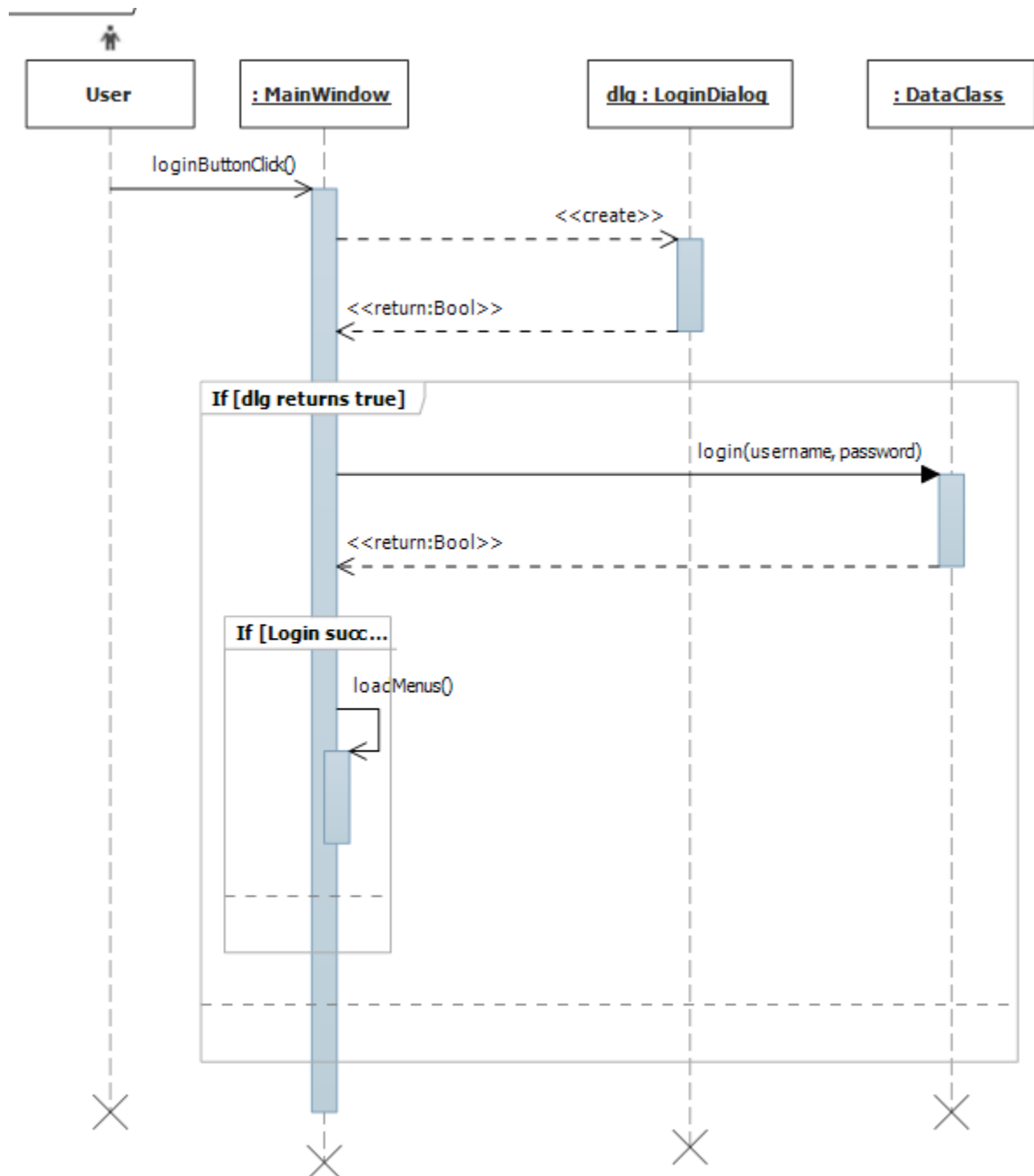


Figure 5: Login sequence diagram

3.2.4 Class diagram

A class diagram showing the most important classes of the system and how they interact with each other is displayed below. The largest of the classes, MainWindow, clearly has the largest number of operations due to the high number of buttons and handlers it must have.

The smallest of the classes, OrderItem, is logically the simplest class since it is very focused on what it alone must do without depending on other classes and therefore is a class with high cohesion.

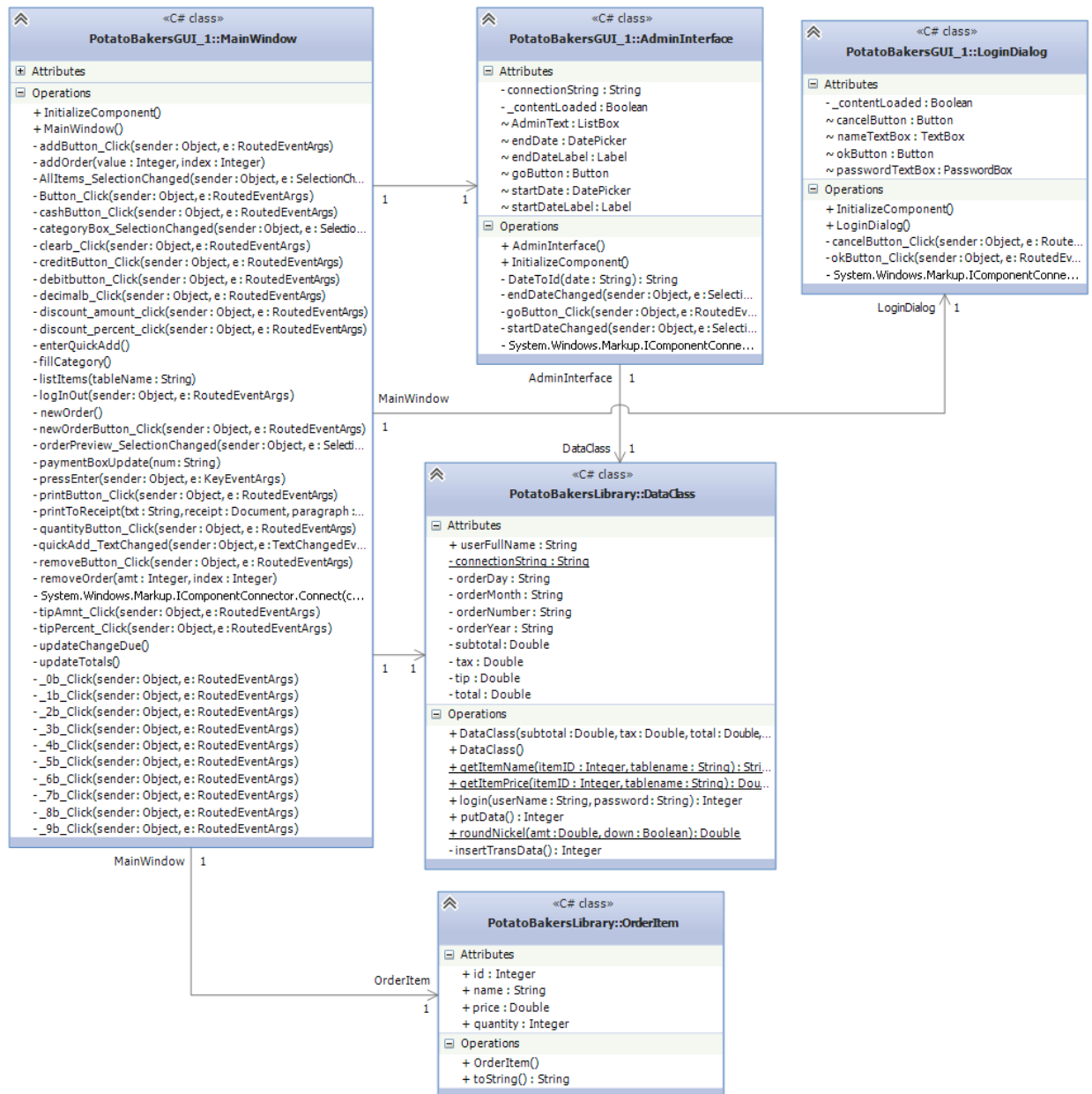


Figure 6: Class Diagram

4. Technical documentation

4.1 Programming language and platform

This section describes the more technical aspects of the software system including its platform, programming language, development tools and database tables and schema.

The software system was written using the programming language C Sharp and created using .NET framework for Windows OS. These were chosen since Windows is a more common operating system in the field of POS computers.

The development tool used was Visual Studio Ultimate 2013. This tool provided the development team the vast majority of all features needed in the development process including a built in compiler and editor, automated testing, connection to the team foundation server, tools for easily designing the user interface and various forms of system modelling.

Some software reuse was introduced with the integration of the open source library “iTextSharp”, a library that allows writing to PDF files.

The tool used for database related implementation was SqlServer 2012. This turned the computer into its own local database that the system could access during runtime.

4.2 Database Table and Schema

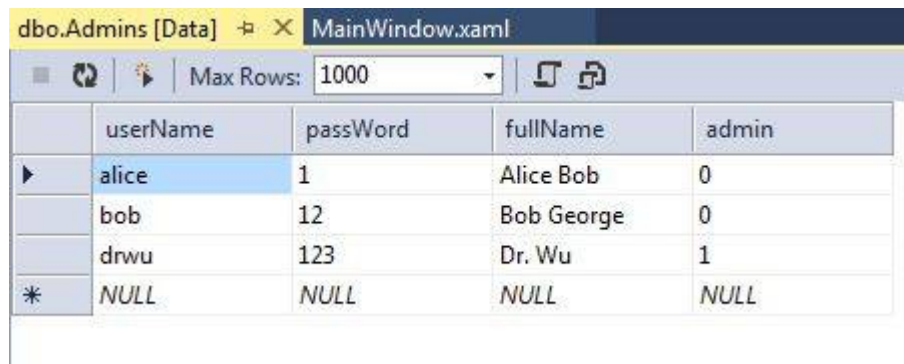
For database management SqlServer has been used. In this restaurant system the layout of the database tables is simple. All the tables are stand-alone; there is no interaction among them. Most tables correspond to menu categories. Keeping the menu stored in a database rather than as part of the system allows for pricing change, addition/removal of menu items, etc. all without having to make any changes to the system itself. Other tables store information about the menu categories themselves, as well as employee information. The following tables are used:

1. Admins
2. Orders
3. Category

4. Appetizers
5. FlamedN
6. Gunkan
7. HandRoll
8. Maki
9. Nigiri
10. Sashimi
11. SpecialityRoll

4.2.1 Admins:

This table stores the Username and Password for the Admin. It also stores admin's full name and an admin field. The field admin indicates who is able to access the admin interface.



	userName	passWord	fullName	admin
▶	alice	1	Alice Bob	0
	bob	12	Bob George	0
	drwu	123	Dr. Wu	1
*	NULL	NULL	NULL	NULL

Figure 7: Employee database table

4.2.2 Orders:

This table is used to store the orders of the restaurant. It has a unique Id for each order. It stores the year, month and day in which the order is placed in the field year, month, and day respectively. The field orderno stores the unique order Id for a particular order. The subtotal, tax, total and tip for an order are stored in the field subtotal, tax, total and tip respectively.

This table is used by the cashier to store the order records and by the admin to calculate the total and average sales of the restaurant for any given period of time.

dbo.Orders [Data] Max Rows: 1000									
	Id	year	month	day	orderno	subtotal	tax	total	tip
	201411115145046	2014	11	15	145046	4.2	0.546	4.746	0
	201411115145050	2014	11	15	145050	8.7	1.131	9.831	0
	201411115145054	2014	11	15	145054	10.69	1.3897	12.0797	0
	201411115145057	2014	11	15	145057	24.62	3.2006	27.8206	0
	201411115145108	2014	11	15	145108	8.7	1.131	9.831	0
	201411115155617	2014	11	15	155617	13.8	1.794	15.594	0
	201411115155654	2014	11	15	155654	13.8	1.794	15.594	0
	201411115155655	2014	11	15	155655	13.8	1.794	15.594	0
	201411115161306	2014	11	15	161306	4.5	0.585	5.085	0
	201411116171355	2014	11	16	171355	66.6	8.658	75.258	0
	201411116200955	2014	11	16	200955	55.8	7.254	113.054	50
	201411116201002	2014	11	16	201002	55.8	7.254	113.054	50
	201411117004219	2014	11	17	4219	12.6	1.638	14.238	2
	20141121110418	2014	11	21	110418	7.1	0.923	8.023	0
	20141124104507	2014	11	24	104507	9.6	1.248	10.848	0
	20141124104812	2014	11	24	104812	11.59	1.5067	13.0967	8
	20141129170516	2014	11	29	170516	90.09	11.7117	101.8017	0
	20141129174246	2014	11	29	174246	16.79	2.1827	18.9727	0
	20141129174501	2014	11	29	174501	25.4	3.302	28.702	0
	20141129175031	2014	11	29	175031	21.2	2.756	23.956	0
	20141129175216	2014	11	29	175216	21.2	2.756	23.956	0
	20141129175403	2014	11	29	175403	21.2	2.756	23.956	0
	20141129180935	2014	11	29	180935	21.2	2.756	23.956	0
	20141129181729	2014	11	29	181729	14.8	1.924	16.724	0
	20141129182102	2014	11	29	182102	12.8	1.664	14.464	0

Figure 8: Order history table

4.2.3 Category:

This table stores menu category of the restaurant. In category field each type of menu is stored and its unique id is stored in Id field. If a new category of the menu is needed, it can be added to the Category table directly.

dbo.Category [Data]		dbo.Orders [Data]
		Max Rows: 1000
	Id	category
▶	0	Appetizers
	1	FlamedN
	2	Gunkan
	3	HandRoll
	4	Maki
	5	Nigiri
	6	Sashimi
	7	SpecialtyRoll
*	NULL	NULL

Figure 9: Category table

Each category has several items which are stored in individual table under every category names. The table construction is same for each category. Therefore only one table construction has been described in following paragraph.

4.2.4 FlamedN (and others):

This table stores the name, price and unique item id of the items in menu category “FlamedN”.

dbo.FlamedN [Data]		dbo.Admins [Data]	MainW
		Max Rows: 1000	
	ItemID	ItemName	price
▶	451	Salmon with Bl...	3.5
	452	Salmon with Sp...	3.5
	453	Tuna with Blac...	3.8
	454	Crab Stick with ...	3.4
	455	Unagi ...	4.2
*	NULL	NULL	NULL

Figure 10: FlamedN Menu table

There are other tables for each menu category such as Appetizers, Gunkan, HandRoll, Maki, Nigiri, Sashimi, SpecialityRoll. All these tables have the same field as Appetizers.

5. User Documentation

Use of the system is divided into sections based on use cases (see requirements document for details). The following sections describe use of the system based on each such use case, including screen shots of the graphical user interface where appropriate.

5.1 Add Item to Order

When user selects a new menu item from the currently opened menu, a string is automatically generated and displayed in the multipurpose input text box. The string is of the form “A<itemid>Q<quantity>”, where A represents the intent to “Add” an item to the order, <itemid> represents the particular item’s unique identifying number, and Q followed by the number <quantity> represents the quantity of that item to add. To complete the addition, the user must press the “Enter” button.

Demonstrated below is the addition of item number 501 (sashimi salad) from the menu “appetizers” with quantity 6 as can be seen from the input string “A501Q6”.

The screenshot shows a POS application window titled 'MainWindow'. It features a menu list on the left, a central display area, and a numeric keypad on the right. A red box highlights the 'Appetizers' category in the menu, and another red box highlights '501 Sashimi Salad' in the items list. A red arrow points from this item to the input string 'A501Q6' in the central display. Another red box highlights the numeric keypad, with arrows indicating the sequence of button presses: '5', '0', '1', 'Q', and '6'. The 'Totals' section on the right shows a subtotal of \$37.20, tax of \$4.84, and a total amount due of \$42.04. The 'Order Preview' section at the bottom left shows '501 Sashimi Salad x6' for a total of 37.2. The 'New Order' button is at the bottom right.

Menu	Items	Totals
Appetizers	501 Sashimi Salad	subtotal: \$37.20
FlamedN	502 Wakana Salad	tax: \$4.84
Gunkan	503 Masago Salad	gratuity: \$0.00
HandRoll	504 Avacado Salad	Amount Due: \$42.04
Maki	505 Edamame	
Nigiri	5061 Miso Soup	
Sashimi	5062 Spicy Miso Soup	
SpecialtyRoll	5073 Crunchy Shrimp	
	5075 Crunchy Shrimp	
	5078 Crunchy Shrimp	

Input String: A501Q6

Order Preview:

Item	Qty	Price
501 Sashimi Salad	x6	37.2

Amt Due \$ 42.04

New Order

Figure 11: Adding items to order

5.2 Remove Item From Order

When a user selects an item from the Order Preview list the system behaves in much the same way as when an item is selected from the menu list except in that the automatically generated string is of the form “R<itemid>Q<quantity>”. In this input string, “R” stands for “remove”, <itemid> is the unique item number of the item to be removed, and Q followed by the number <quantity> represents how many of that item are to be removed. To complete the removal of the item from the order, the user must press the “Enter” button.

Demonstrated below is the removal of item number 503 (Masago salad) from the order preview with quantity 1 as can be seen from the input string “R503Q1”.

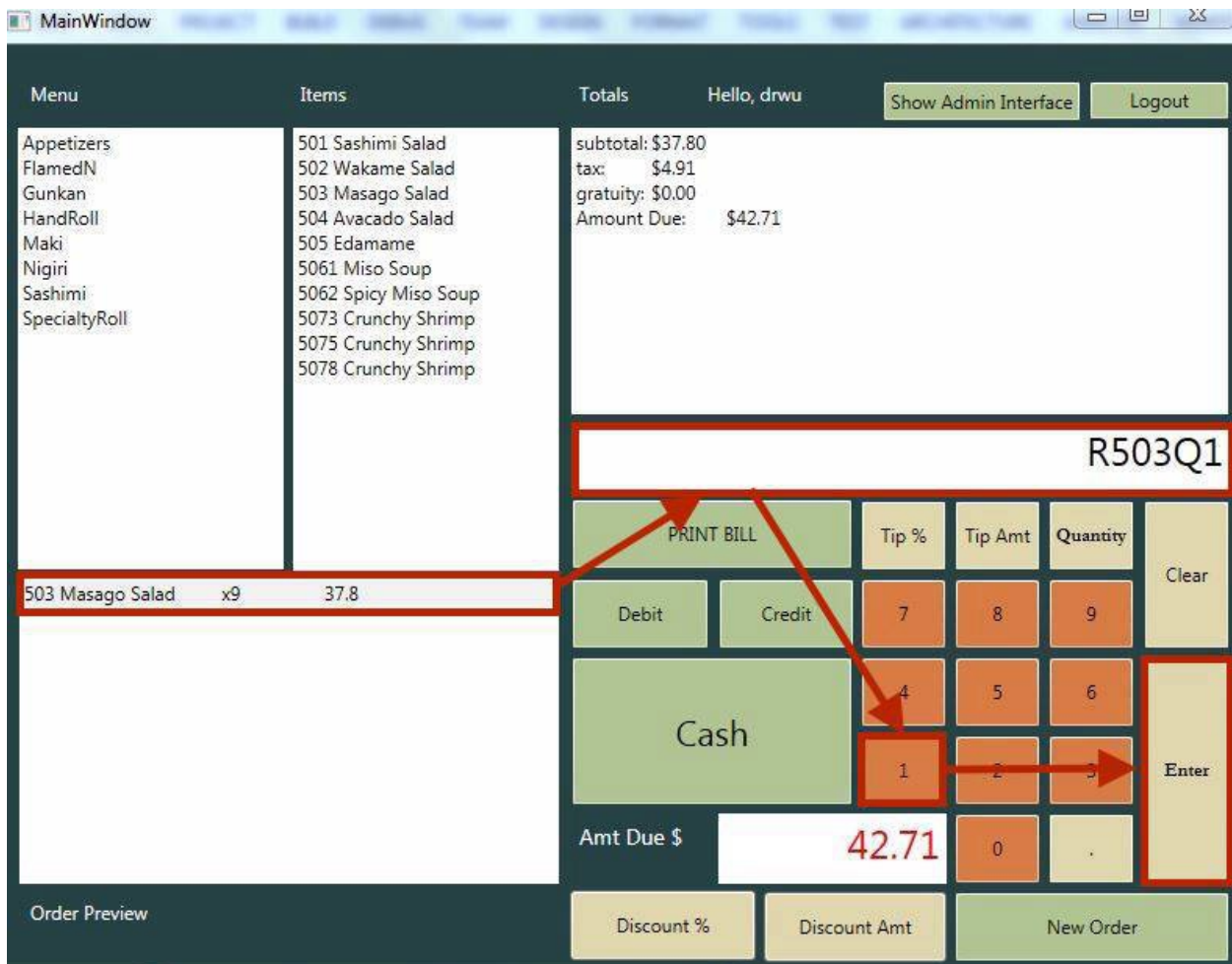


Figure 12: Removing items from order

5.3 Print bill - Print Receipt

A bill may be printed and brought to the customer for review before the payment is made. The transaction will not be entered into the database until the final payment is made and the receipt printed. The bill must be printed before any payment is made. To print the bill, the user must press the “Print Bill” button located under the totals box. The bill will be printed to a pdf file called “bill.pdf” and will show the basic contact info for the restaurant and a breakdown of all costs associated with the current order as well as the total amount due. Once the payment is made in full, the number in the “amount due” box will change to green, and the “Print Bill” button will become the “Print Receipt” button, indicating that the payment is complete. Pressing the “Print Receipt” button will print receipts to two pdf files: “customerVersion.pdf” and “merchantVersion.pdf”. Additionally, once the “Print Receipt” button has been pressed, the current transaction will be stored to the database.

5.4 Make Payment

The screenshot shows a restaurant POS interface with the following components:

- Menu:** A list of food items including Appetizers, FlamedN, Gunkan, HandRoll, Maki, Nigiri, Sashimi, and SpecialtyRoll.
- Items:** A list of specific items with quantities and prices, such as 501 Sashimi Salad (x6, 37.2).
- Totals:** A summary of the bill showing subtotal (\$37.20), tax (\$4.84), tip (\$0.00), round nickel (\$0.01), and amount due (\$42.05). It also shows 'Payment Made: \$80.00' and 'Owe Customer: \$37.95'.
- Payment Keypad:** A grid of buttons for payment methods (PRINT BILL, Debit, Credit, Cash) and a numeric keypad (0-9, ., /, *, %, Enter, Clear).

Red arrows highlight the payment process: one arrow points from the 'Payment Made' field to the 'Cash' button, another points from the 'Cash' button to the '80' input field, and a third points from the '80' input field to the 'Enter' button.

Figure 13: Making a payment

There are three payment methods made available to customers: Cash, credit and debit. In order to pay using cash, the user must first enter the dollar amount to be paid into the input box using the number key pad and then press the large button labeled “Cash”. Rounding to eliminate the need to pay with pennies is eliminated in the case of paying the full amount with cash. However, if the client decides to pay credit or debit, the user simply presses the “credit” or “debit” button respectively and the full and exact amount will be deducted from the total. In order to pay with different methods, the customer must make the first payment in cash, and the rest can be paid using debit or credit. Customers cannot make partial payments using debit or credit.

5.5 Split Bill

To split the bill the user must treat each customer as if they had ordered separate bills, and apply discounts to any shared items. For example, if person A orders Item C and person B orders item D and both decide to split the price of item E, then the bill for person A would contain items C and E, with a discount equivalent to half the price of item E. Similarly, the bill for person B would contain items D and E, with a discount equivalent to half the price of Item E.

5.6 Add Tip

The screenshot shows a restaurant POS interface with the following components:

- Menu:** Appetizers, FlamedN, Gunkan, HandRoll, Maki, Nigiri, Sashimi, SpecialtyRoll.
- Items:** 501 Sashimi Salad, 502 Wakame Salad, 503 Masago Salad, 504 Avacado Salad, 505 Edamame, 5061 Miso Soup, 5062 Spicy Miso Soup, 5073 Crunchy Shrimp, 5075 Crunchy Shrimp, 5078 Crunchy Shrimp.
- Totals:** subtotal: \$13.20, tax: \$1.72, gratuity: \$1.98, Amount Due: \$16.90.
- Input Fields:** A numeric keypad with buttons for digits 0-9, a decimal point, and a 'Clear' button. A 'Tip %' button is also present.
- Buttons:** PRINT BILL, Debit, Credit, Cash, Enter, Discount %, Discount Amt, New Order, Show Admin Interface, Logout.

Red annotations indicate the process of adding a tip:

- A red box highlights the 'Amount Due: \$16.90' in the Totals section.
- A red box highlights the input field containing '15'.
- A red arrow points from the 'Tip %' button to the input field.
- A red arrow points from the 'Cash' button to the 'Enter' button.

Figure 14: Adding a tip

To add a tip to a payment, the user has the option to apply a percentage or a dollar amount. If using the tip percentage, the user must enter the desired percentage into the input box as a whole number (ex. "15" for 15%), then press the "Tip %" button. If the tip is a particular

amount, then the user must enter the dollar amount of the tip into the input box and then press the “Tip Amt” button. Once a payment has been made in full, tips may no longer be added.

5.7 Add Discount

The screenshot shows a POS system interface with the following sections:

- Menu:** Appetizers, FlamedN, Gunkan, HandRoll, Maki, Nigiri, Sashimi, SpecialtyRoll.
- Items:** 501 Sashimi Salad, 502 Wakame Salad, 503 Masago Salad, 504 Avacado Salad, 505 Edamame, 5061 Miso Soup, 5062 Spicy Miso Soup, 5073 Crunchy Shrimp, 5075 Crunchy Shrimp, 5078 Crunchy Shrimp.
- Totals:** subtotal: \$13.20, tax: \$1.72, gratuity: \$1.98, Discount: -\$3.00, Amount Due: \$13.90.
- Buttons:** PRINT BILL, Tip %, Tip Amt, Quantity, Debit, Credit, Cash, Enter, Discount %, Discount Amt, New Order.
- Input Fields:** 3 (quantity), 13.90 (Amount Due), 13.90 (Discount Amt).

Figure 15: Adding a discount

Discounts work much in the same way as tips; the user has the option to apply a discount as a percentage or a dollar amount. If using the discount percentage, the user must enter the desired percentage into the input box as a whole number (ex. “50” for 50%), then press the “Discount %” button. If the discount is a particular amount (as in the case of splitting the bill), then the user must enter the dollar amount of the discount into the input box and then press

the “Discount Amt” button. Once a payment has been made in full, discounts may no longer be applied.

5.8 Login - Logout

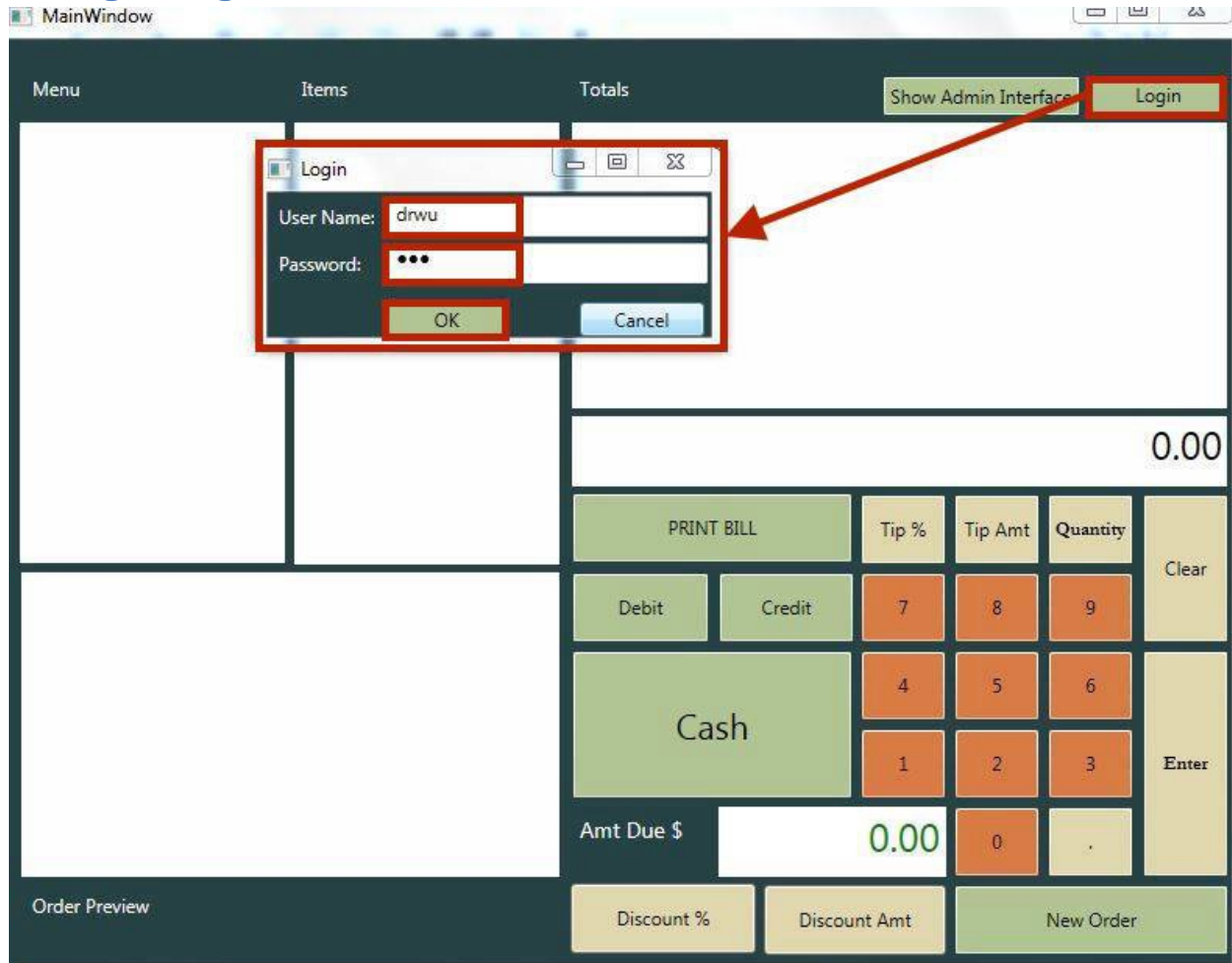


Figure 16: Logging in

In order to use the system the user must be logged in. This is done by pressing the “Login” button in the upper right corner of the main window, after which a login window will pop up. The user then must fill out his or her user name and password into the appropriate text boxes and press “OK”. If the name and password match, the menus are loaded, a welcome message is displayed on top of the totals list box, and the “Login” button becomes a “Logout” button. To log out, the user must click the “Logout” button.

5.9 Admin Interface (get range of orders)

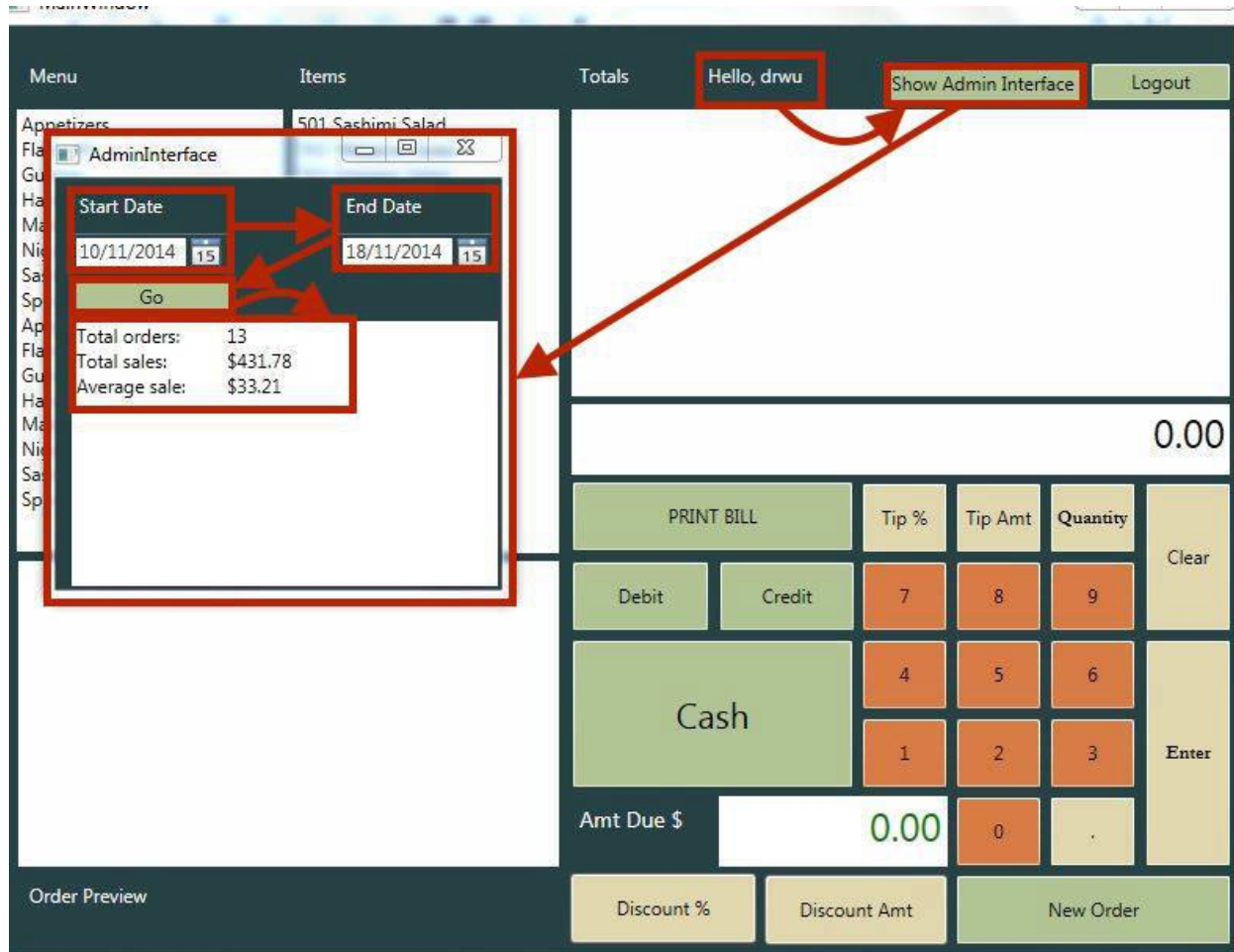


Figure 17: Using the admin interface

If a user with admin privileges is logged in he or she may use the “admin interface”; a small sub-program that allows an administrator to retrieve information about sales between an input interval of time. To retrieve the information the user simply must enter the start date and end date and then press “Go”. The information will appear in the list box below the “Go” button.

5.10 New Order

To create a new order, or reset the current one, the user only has to click the “New Order” button located at the bottom right of the main window. Pressing this button will reset all currently stored information about the current bill or purchase, and clear the totals list box and the order preview list box.

6. Software Testing

6.1 Overview of testing

The majority of tests during each iteration were done informally, though some formal testing was done during the final iteration. Each testing session would be done usually after implementing a new feature, in order to test the effectiveness of the implementation. If the feature were found to be lacking, development would begin again, a new implementation would be made, and then testing would be done once more. This process was repeated until the feature in question was in a considerably decent state and deemed acceptable by the tester.

6.2 Regression testing

Sometimes when implementing bug fixes or even when adding new features, parts of the system that were deemed to be in acceptable condition would no longer work. Because of this, much regression testing was done. Each time a new feature was implemented, other features would be tested, even though they were not the features in question. This helped identify hidden bugs that might have been overlooked otherwise.

6.3 Unit testing

Some unit tests were made to test individual operations of classes. These tests were automated with the help of visual studio's built-in testing suite. Automating these tests was extremely useful since many tests could be run in fractions of seconds and regression testing was made easier as a result.

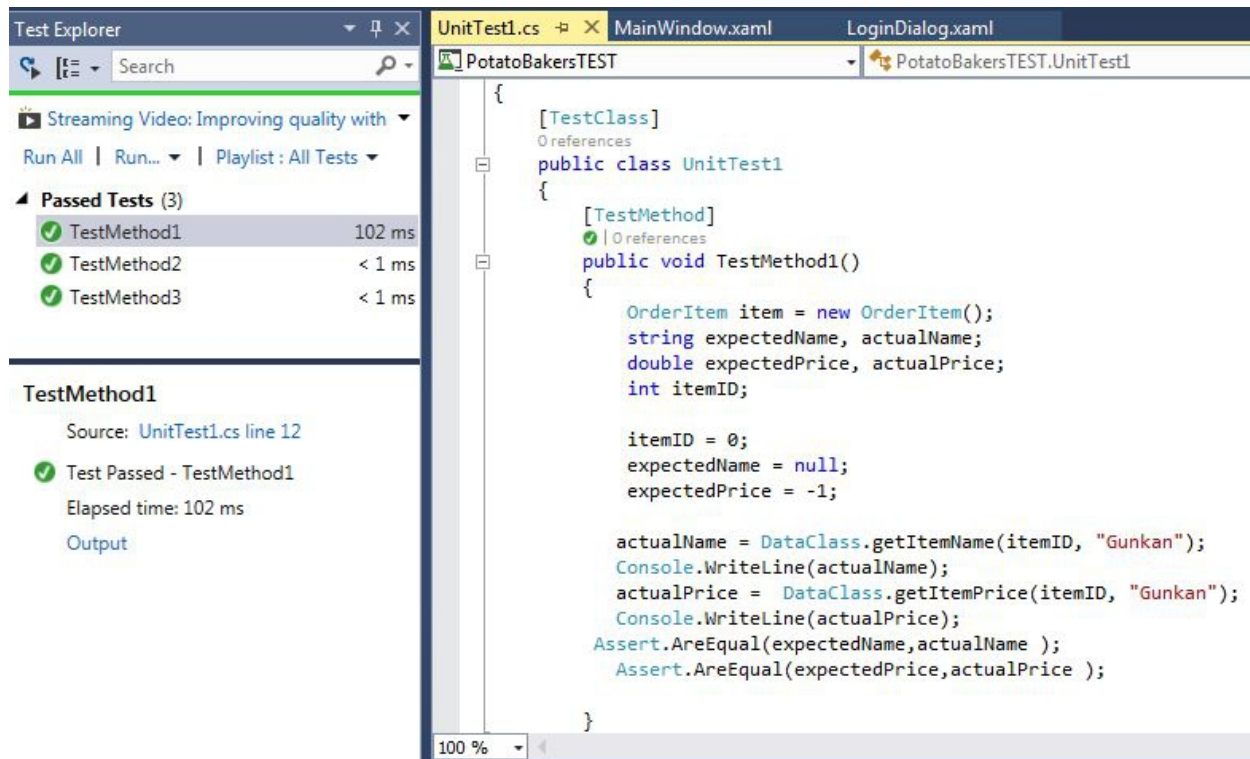


Figure 18: Unit testing using Visual Studio

6.4 Software inspection and validation

Doing informal software inspections rather than tests helped identify areas of the software that needed improvement or were not relevant to the client's needs. If the system was found to be lacking a feature that was deemed necessary by the inspectors, but not explicitly requested by the client, that feature was usually added. This helped with system validation and maintaining relevance to the client's needs.

6.5 Release testing

The Software has currently been handed to a separate testing team with a usage document in order to further test the system. In the hands of a team that did not develop the software, it may be easier to identify bugs and other oversights that the developers might have missed, even with their own test cases. Feedback from this testing cycle significantly contributes to the improvement of software quality.

Summary of total work done

Iteration 1:

	Angana (%)	Caitlin (%)	Josh (%)	Halima (%)	Salyha (%)
Menu Choice (1%)	90			10	
System Design (5%)	20	20	20	20	20
GUI Design (5%)	20	20	20	20	20
Team Foundation Server setup (3%)		100			
Database design (5%)	40	20		40	
GUI Implementation (10%)	10	45	45		
GUI Interaction Logic (20%)		20	80		
Database Implementation (20%)		90		10	
Table data entry (5%)	100				
Database interaction processes (15%)		90	10		
Refactoring (8%)			100		
Bookkeeping (2%)	50				50
Iteration Report (1%)	12	52	12	12	12

Iteration 2

	Angana (%)	Caitlin (%)	Josh (%)	Halima (%)	Salyha (%)
Database transaction entry code (20%)				100	
Database transaction entry implementation (20%)		34		33	33
Quick-add (5%)			90		

			(incomplete)		
Refactoring (5%)		50	50		
Redmine/bookkeeping (10%)			5	10	85
Tip feature implementation (10%)		50			50
Payment feature implementation (10%)		50			50
Testing research (5%)	50 (incomplete)				
Print to file feature (10%)		30	20	20	30
Iteration report (5%)	5	65	5	10	10

Iteration 3

	Angana (%)	Caitlin (%)	Josh (%)	Halima (%)	Salyha (%)
Cash rounding (10%)	5	30	35		30
GUI redesign (15%)	35		35		30
Interface simplification (10%)		90	10		
New Bill (5%)		5	95		
Admin Login (5%)		100			
Admin data retrieval (10%)			50	50	
Admin interface (5%)		50		50	
Testing (10%)	50	50			
Payment feature (10%)	5	5	80		10
Presentation(10%)	25	25	25		25
Bookkeeping (5%)					100
Iteration report (5%)	50	50			

Final Iteration

	Angana(%)	Caitlin(%)	Josh(%)	Halima(%)	Salyha(%)
New Order Button (15%)		100			
Customer and merchant copy of receipts (15%)		100			
Admin interface adjustments (15%)			100		
Discount buttons (15%)		85	15		
Book keeping and documentation(5%)					100
Employee Login(15%)		15	85		
Admin permissions(20%)		70	30		

Final Report

	Angana(%)	Caitlin(%)	Josh(%)	Halima(%)	Salyha(%)
Introduction (14%)	50				50
Software Requirements (14%)		100			
Design Specification (14%)		100			
Technical Documentation (14%)		10		90	
User Documentation (14%)		100			
Software Test Plan (14%)		90	10		
Appendix (Testing) (16%)			100		

8. References

Restaurant: <http://www.sushi-one-express.com/>

Visual Studio server: <https://facchinc.visualstudio.com/>

Redmine: <https://redmine.cs.uwindsor.ca/>

Software Engineering: 9th Edition, Ian Sommerville, 2010