# Introduction

## 4.1. Project Background

The project to be completed is to develop a fully functioning meal debit system (hereinafter referred to as the System). The System is to be installed and utilised within the vicinity of a dining service provider (with some exceptions). Within the project, The System will be constructed so that it should be able to be installed and utilised by any dining service provider, that is to say, that minimum dependencies of external data should be expected. However, for the sake of this documentation, The System will be constructed with ‘being used within the cafeteria of a school environment’ in mind.

One feature of the System is that it utilises prepaid cashless paying to increase security and reduce risk. To make a transaction, the customer must have an account registered within the System, and have reload balance associated with the account, which can be done through the browser. The cashier enters the products purchased into the register, and the amount to be paid is displayed. The customer can then enter the amount to be paid into their device, which will generate a barcode for the cashier to scan. The barcode will contain the customer ID and the amount to be deducted from the account balance within the database.

## 4.2. Project Objectives

The purpose of the project is to construct a fully-functioning meal debit system with minimum data dependencies to function properly. The system should be able to handle all transactions being done in the dining service provider with minimal exceptions. The software of the system is expected to handle cashless transactions, as well as being able to handle management of associated parts of the system, such as managing employees, etc. It can solve the problem of one crucial problem that most cashless meal debit systems meet: The reloading process is too much of a pain since they have to do it in person and in cash, and they eventually abandon the system and go with cash again. What the system does is provide a convenient way to perform balance reloading through online banking, and reduces extra components such as the card, because they can use their devices to perform cashless payment instead of being assigned another card.

The System will be constructed of two parts: Part A and Part B. Part A is the desktop application that will be installed into the device that the cashier uses, and will be where the point-of-sale functions execute. Part B will be a webpage that will be designed to perform best on a mobile browser, that will allow customers to perform functions such as online top-up, view transaction history, etc. Note that Part A should be used by the employees only and Part B, customers only. If an employee decides to use the system as a customer, they should do so using Part B, that is to say, the functions between parts A and B should not mingle with each other.

## 4.3. Project Scope

This system is designed to be used by novice level users. Most of the functions that the users (customers and employees) can perform will be simplified as much as possible, and they do not require the users to have technical knowledge to perform. The direction of the design will try to incline towards ‘intuitive’ as much as possible.

The main users of the System are the customers and the employees, which are divided into cashiers and managers. The customers can access the System to check their account balance and reload using online banking; the cashiers can access the system while at work to use the point-of-sale functions and to manage transactions; and the managers can manage the menu items, and manage employee data.

## 4.4. Tools Used in the Project

### 4.4.1. Electron.js

Electron.js (hereinafter referred to as Electron) is an open-source framework developed and maintained by Github. It is used to develop desktop software using development language intended for web applications, such as HTML, CSS, and JavaScript. Today, many widely-used applications were developed by Electron, including Atom (text editor), Visual Studio Code (text editor), Discord (voice chat application), and Slack (online collaboration tool). (Electron, 2019) The reason Electron is chosen in the project is because the team members are fairly proficient in the languages mentioned, and believe that a desktop application is more acceptable to be used in a point-of-sale system than a web-based software, which depends on a browser and can be accessed regardless of location.

# Use Case Diagram



Figure

# Final Screen Designs

## Part A

### Home Page

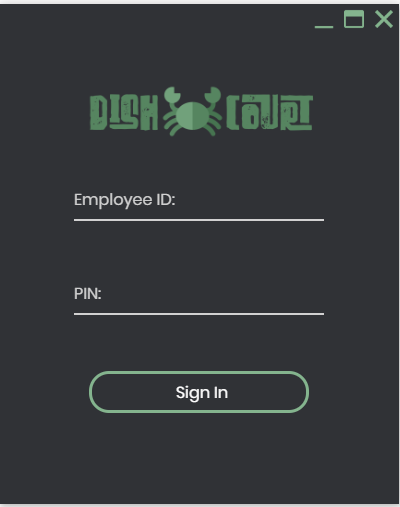


Figure 24 Log In Screen

The figure above shows the home page of Part A system. It only consists of basic operation buttons such as “Close”, “Maximize” and “Minimize”. There are also several input fields for user to log in into the system, namely Employee ID, PIN and a sign in button.

### Managerial-Level Employee Page

#### 13.1.2.1 Landing Page

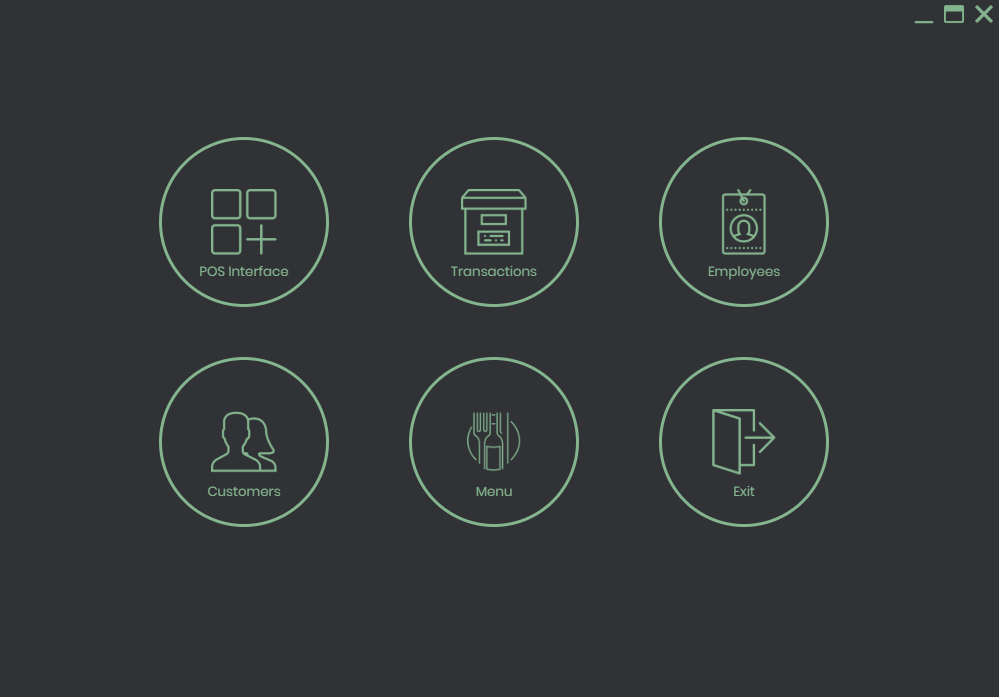


Figure 25 Managerial Level User Landing Page

If the account is logged in as a managerial level employee, this is what will be presented to the user. Managerial level employee gains more right compared to non-managerial levels such as managing employee, managing customers, managing menu and generate report.

#### 13.1.2.2 POS Interface

Figure 26: POS Interface

Figure above shows the POS Interface. It is a simple interface, but it contains everything it needs for this system to record sales. In our system, we use QR code as the main input method to insert data into the system. Thus, this page is designed to be worked without user touching mouse and keyboard. By scanning the QR Codes on the menu paper, user can quickly insert all the data into the system. The upper right text field allows user to scan both item codes and payment codes issued by customer.

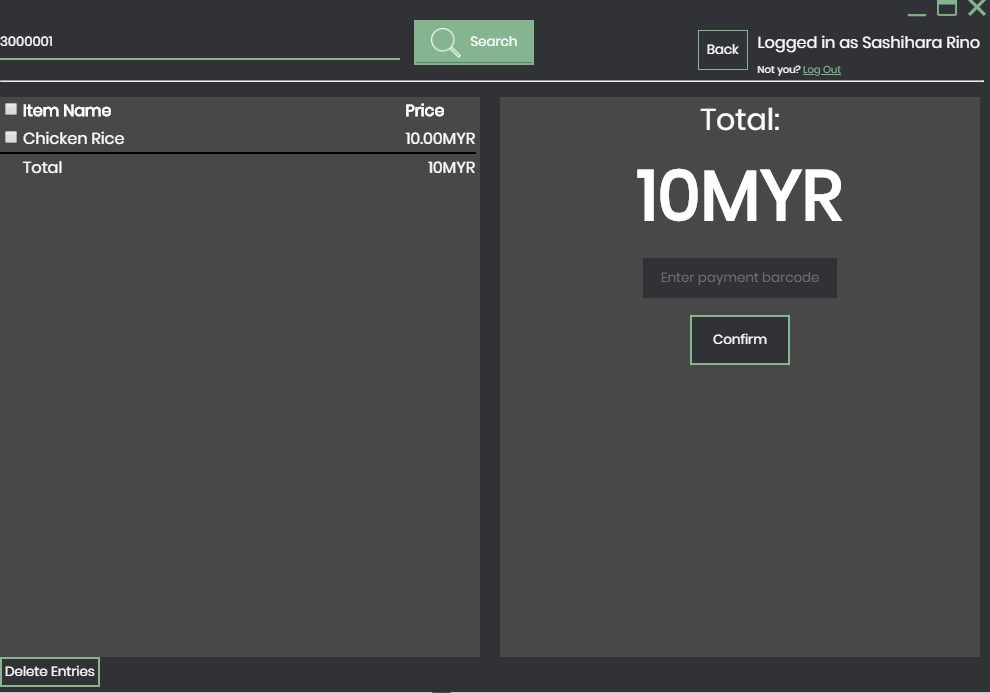


Figure 27: POS Interface with items

After entering a barcode, the item will be added into the table below. The total amount will also be updated, large enough for user to show their customer how much to pay.

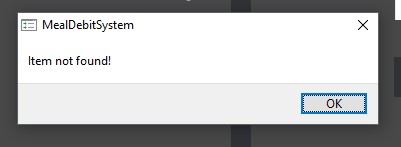


Figure 28: Item not found

In case if the entered item does not exists, a notification will pop up, notifying the user.



Figure 29: Delete item entries

Users can also delete wrong item entry by ticking faulty lines and click “Delete Entries”. In case where everything needs to be deleted, user can tick the checkbox at the upper left to select all and delete.

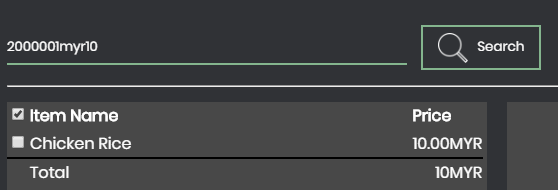


Figure 30: Barcode Bar

Since we want to minimize the use of keyboard and mouse when using the POS system, we allow payment barcodes to be scanner into the item barcode field. Our special algorithm will determine whether entered code is an item code or payment code. If the entered code is payment code, the payment process will go on.

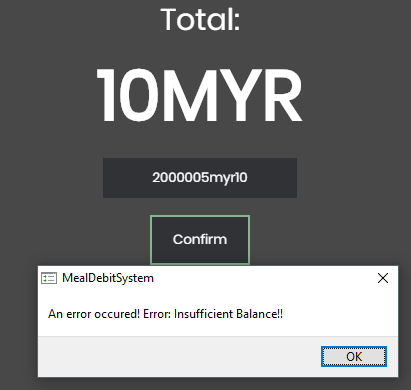


Figure 31: Insufficient balance error

In case where the customer does not have enough balance, an error message will prompt the user.

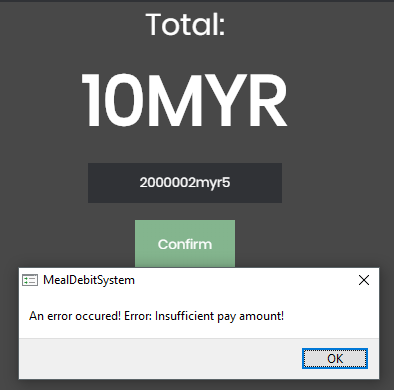


Figure 32: Insufficient balance error

In case where the pay amount is insufficient, user will get prompted as well.

#### 13.1.2.3 Transaction Navigation Page



Figure 33: Transaction Navigation Page

This page contains 3 buttons, namely Report, Transaction and Back. Report button allows user to check report of the business, while Transaction allows user to check a list of transactions that is recorded.

#### 13.1.2.4 Report Page

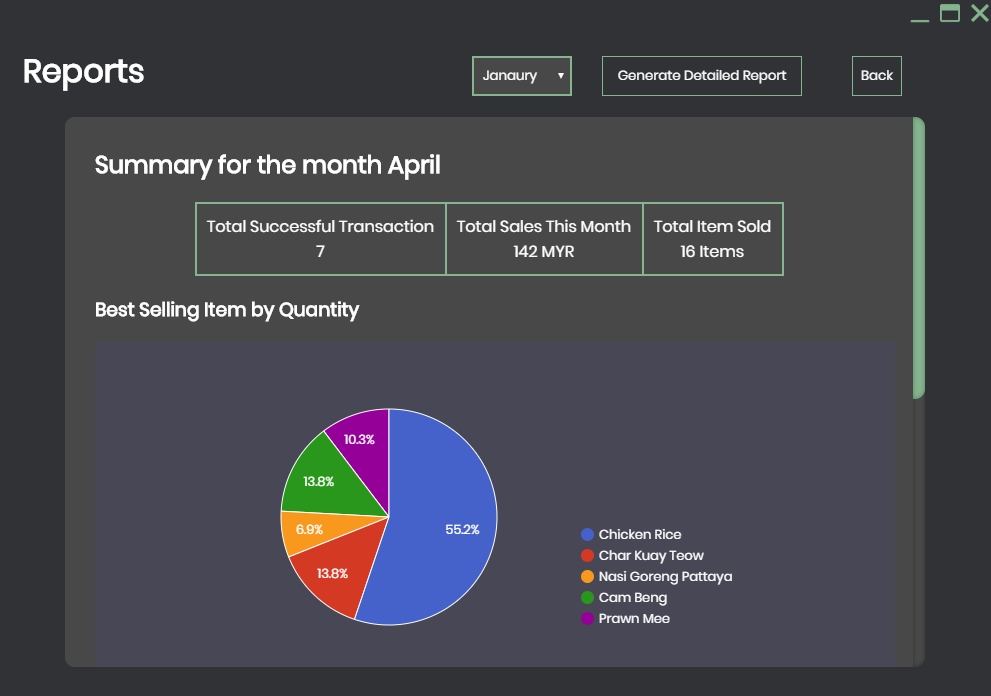


Figure 34: Report Page

Figure above shows the report page. It contains a very brief information and graphs for user to see. User can also generate a much more detailed report on the above controls. To read more about report, please refer to

#### 13.1.2.5 Transaction History Page

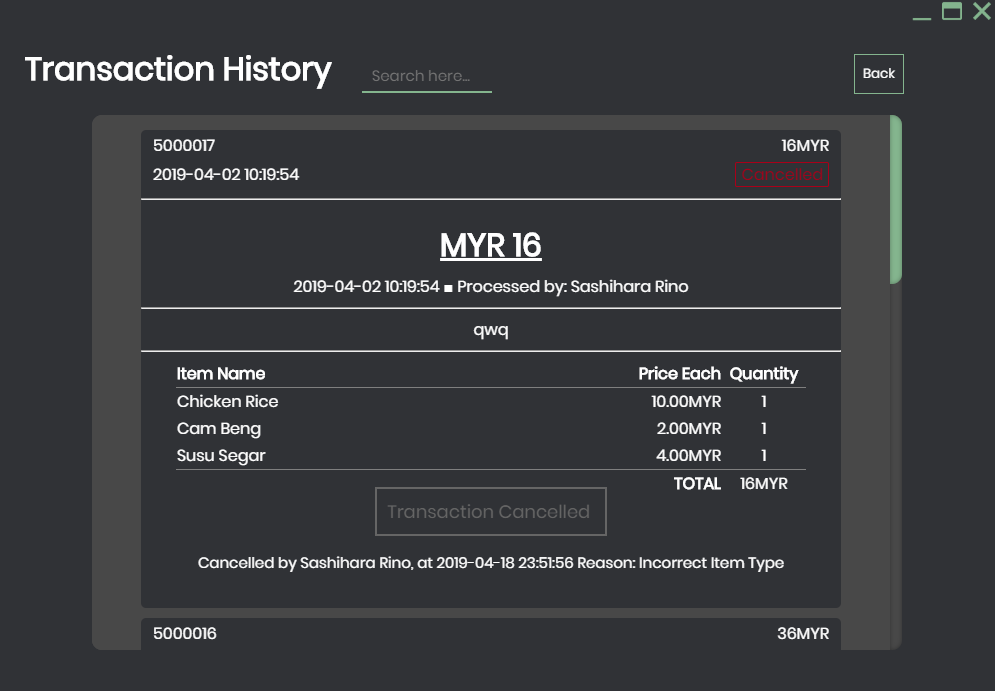


Figure 35: Transaction history page

The transaction history page allows user to see past transactions. User can click onto any transactions they want to see. It is sorted by transaction ID. User can also search for particular transaction with the search bar on top. In this page, after expanding the transaction, user can cancel a transaction in case something went wrong in that transaction.

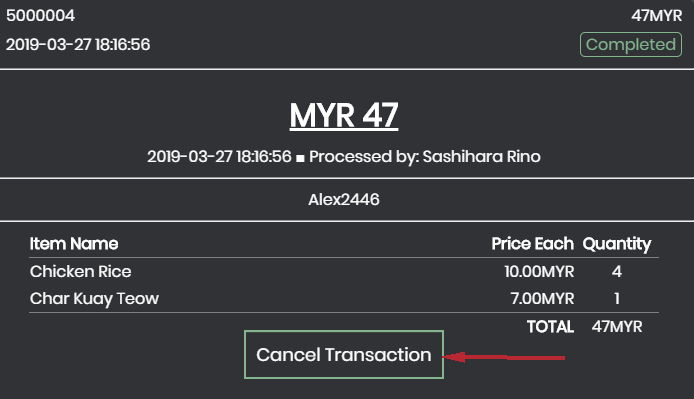


Figure 36: Cancel transaction

To cancel a transaction, user will have to first find the faulty transaction entry and click the “Cancel Transaction” button.

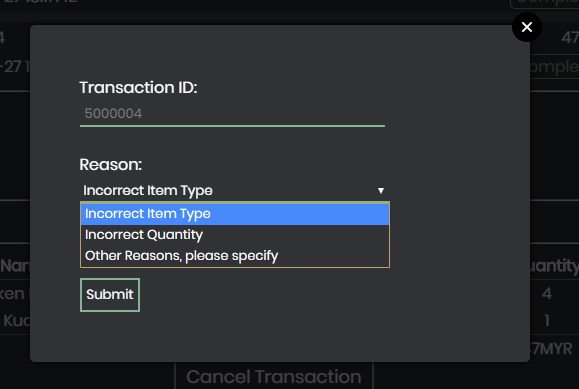


Figure 37: Transaction reason

A modal will pop up to let user choose the reason why the transaction is cancelling. Managerial Level user will be able to cancel any transactions, while non managerial level user requires managerial level user’s permission to cancel any transaction that is older than 1 day.

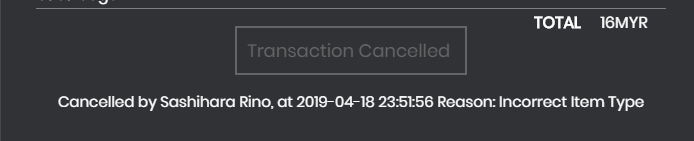


Figure 38: Cancelled Transaction

The cancelled transactions will have reasons stated below the transaction details box.

#### 13.1.2.6 Manage Employee Page

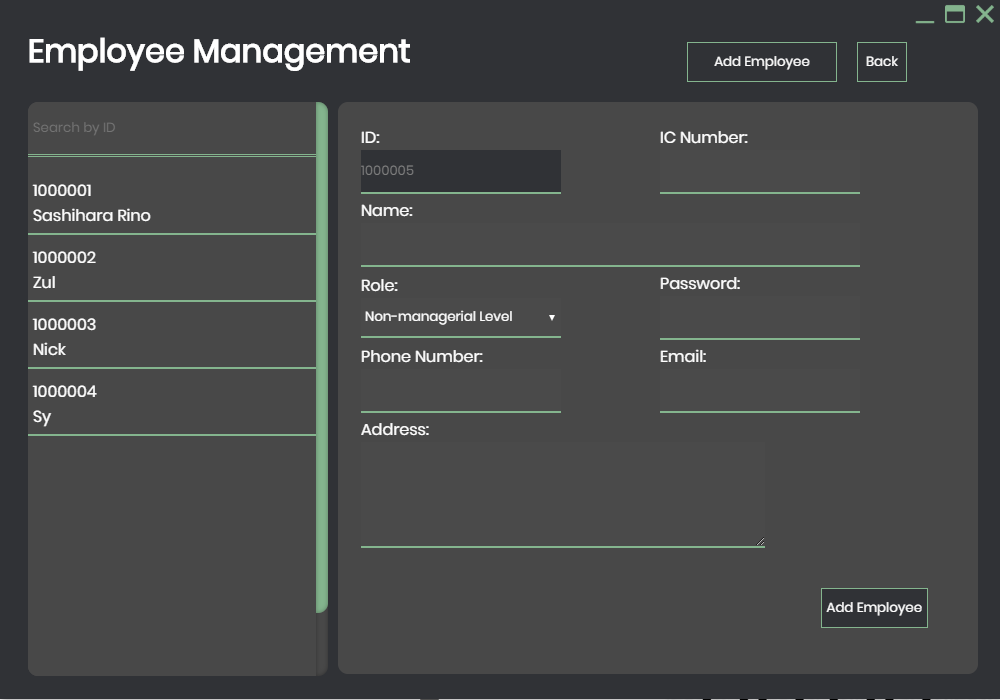


Figure 39: Employee management page

Figure above shows the employee management page. User can just add employee from the right panel.

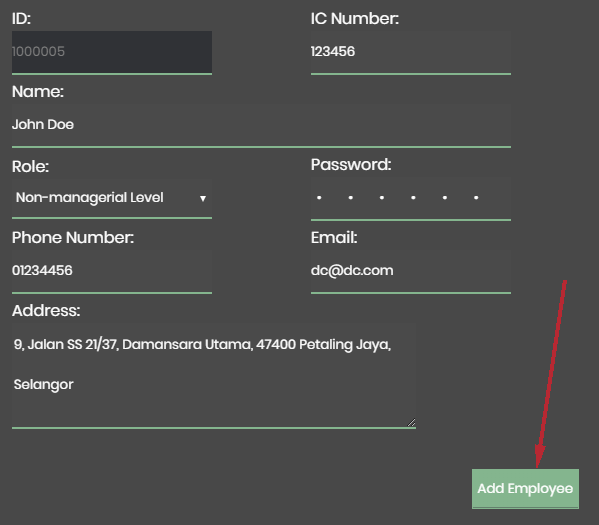


Figure 40: Add Employee

To add an employee, user only has to fill in all the details, tap “Add Employee” and the employee entry will be added.

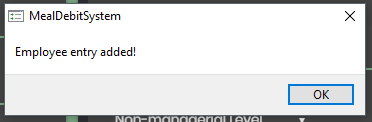


Figure 41: Employee Added Message

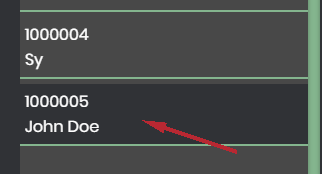


Figure 42: Employee Name and ID

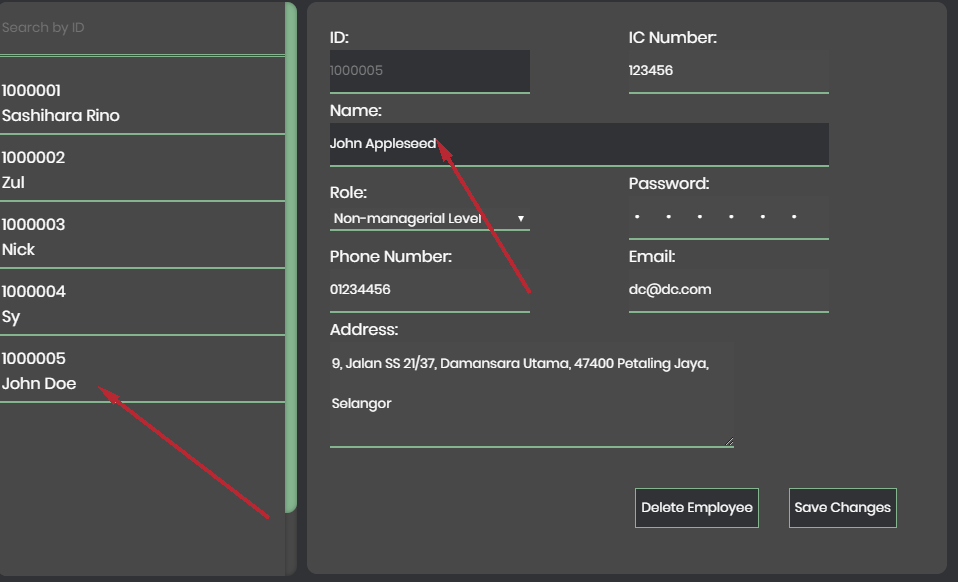


Figure 43: Edit employee info

To edit employee info, user simply has to click the entry that user wish to edit. All the details will be populated at the right panel. User can change what they wish to change here and click “Save Changes”. In this case, we’ll be changing the name “John Doe’ to “John Appleseed”

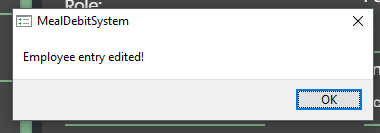


Figure 44: Employee edited message box



Figure 45: Employee id and Name

#### 13.1.2.7 Manage Customer Page

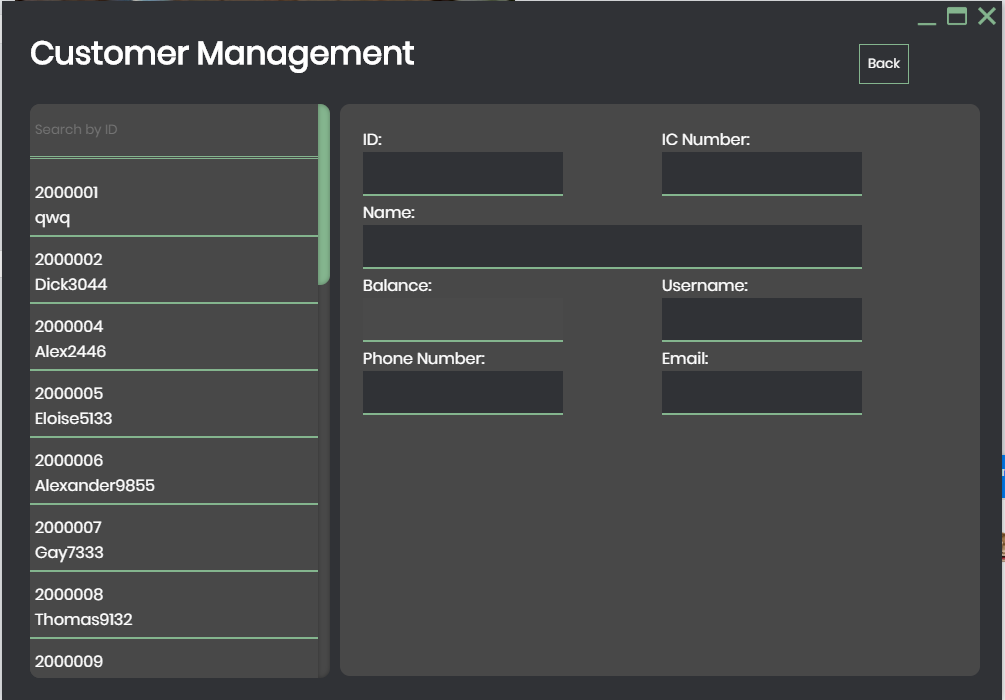


Figure 46: Customer management page

Figure above shows the customer management page. Unlike employee management page, user cannot add customer here. Customer has to create their own account via the online portal (Part B). However, user can click into any user with the left panel and change their balance. This is done just in case where the balance is not updated after user paid, or to delete account.

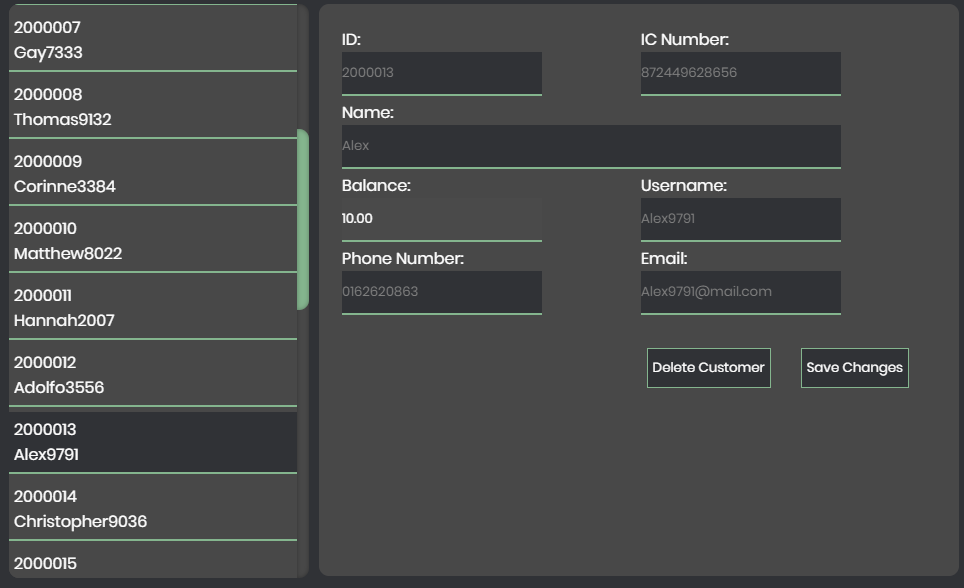


Figure 47: Customer detail

After clicking on a user, all the details other than balance are greyed out. These details can only be changed at customer’s online account. The reason why only balance can be is because that in case where the balance is not updated after user top-up and new balance is not updated to their account, user can quickly solve the problem by crediting money into their account.

There is another function in this page, namely “Delete Customer”. To delete customer, it has to first be requested by the customer. Once the user acknowledges the request, user should arrange a refund to the customer before closing their account. Thus, it is required their account to have 0.00MYR balance left before delete customer action can be carried.

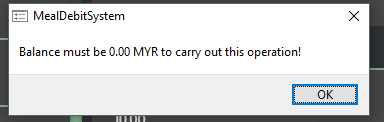


Figure 48: Balance must be RM0 error message box

If the balance is set to 0, the delete customer action will work just fine.

#### 13.1.2.8 Manage Menu Page

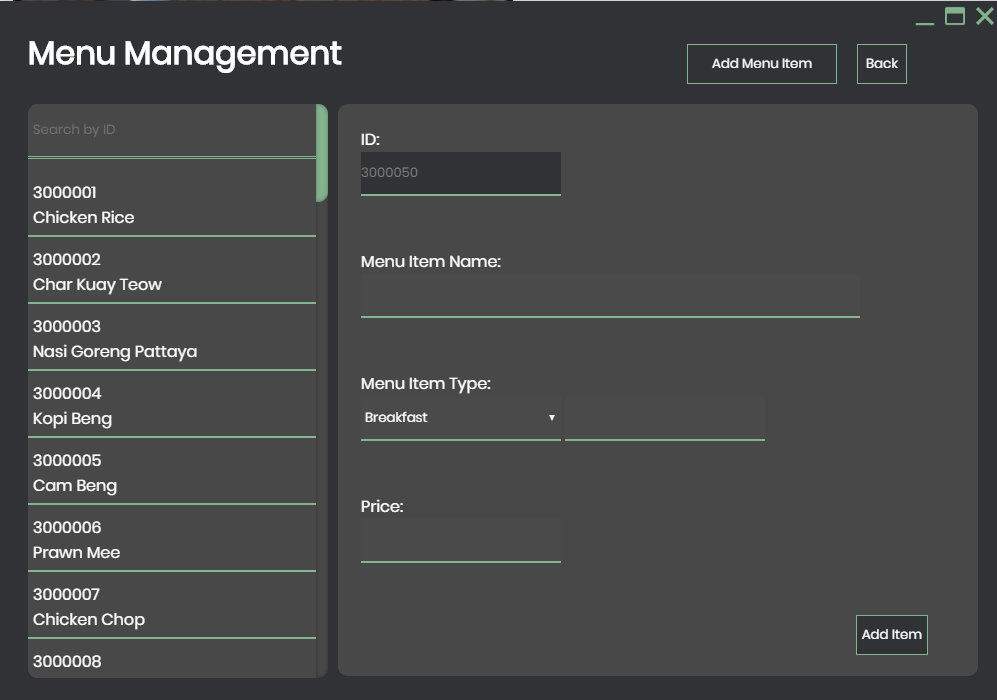


Figure 49: Menu management page

Figure above shows the menu management page. Just like the employee management, user is allowed to insert new menu items in this page.

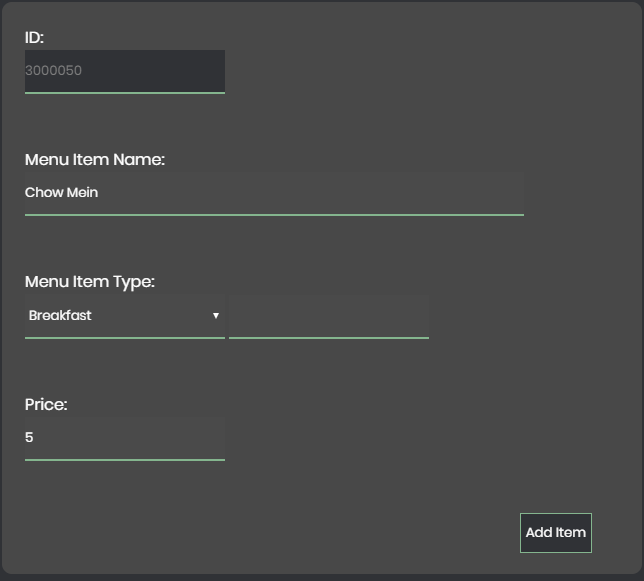
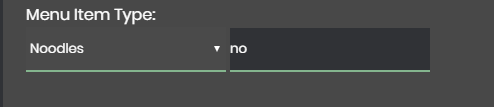


Figure 50: Menu detail

To add a menu, user has to fill in Menu Item name and Price. In the menu item type field, user can type the category in. If a category exists, it’ll suggest user the existing category at the left. If user doesn’t want to use the suggested category, user can create new category simply by ignoring the suggestion; If the user wants to use the suggested category, user can just click the option box, the value will be transferred to the textbox.



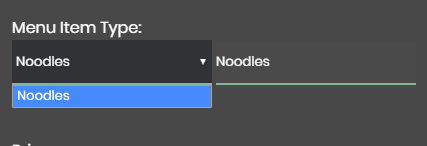
Figure 52: Menu type

Figure 51: Menu type drop down box

### 13.1.3 Non-Managerial Employee Page



Figure 53: Normal employee page

As for non-managerial user, he or she will only be presented with this page, with limited function. The POS Interface is the same as described above (*refer to section 13.1.2.2).*

#### 13.1.3.1 Transaction Page

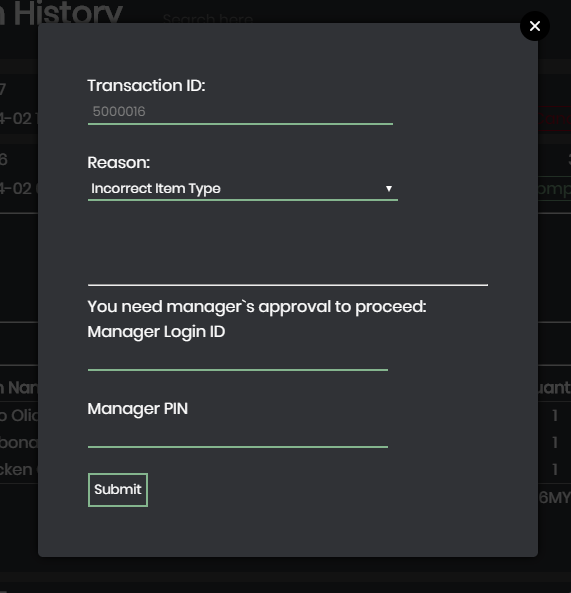


Figure 54: Transaction page

In transaction page, it is mostly the same as described above (*refer to section 13.1.2.5).*

However, for a non-managerial user to cancel a transaction that is older than one (1) day old, user will require manager’s Login ID and PIN in order to proceed. This is limitation is set to prevent non-managerial user to carry out unauthorized cancellations and cause problems in reports.

## Part B

### Home Page

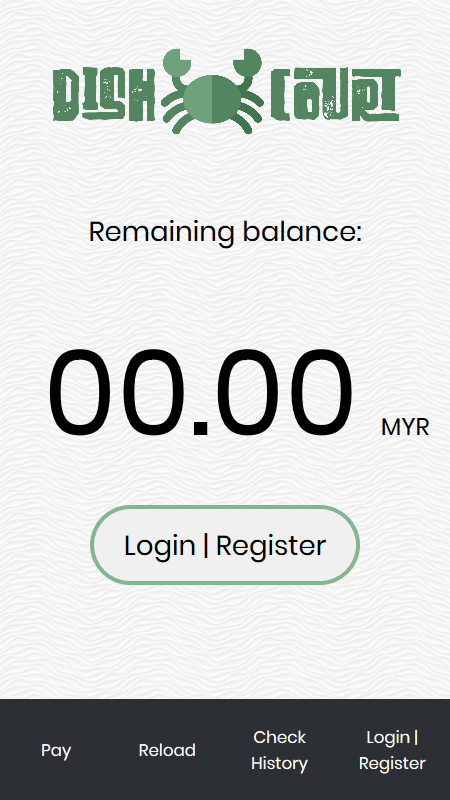


Figure 55

The figure above shows the home page of part B without logging in. Without logging in, all the buttons shown on the screen brings the user to the Login/Register page, where the user will be prompted to log in or to register an account. (See Section 13.2.2.)

### Login/Register Page

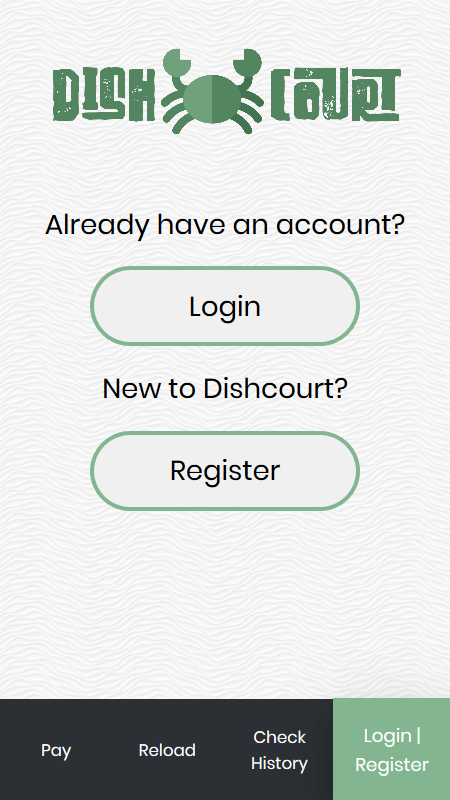


Figure 56

The figure shows the Login/Register page. This page allows the user to navigate to the Login page or the Register page.

### Login

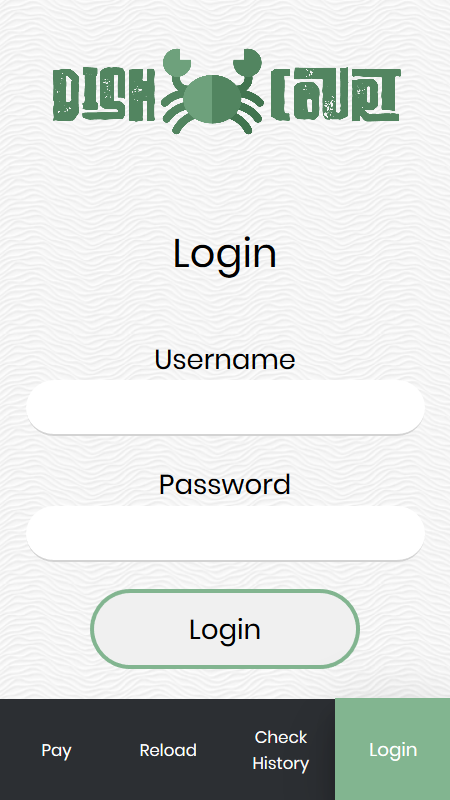


Figure 57

The above figure shows the Login page. Users (customers) can log into the system through this page if an account was created and is available. A user logs in by filling in the credentials and clicking the Login button.

### Home Page (Logged In)

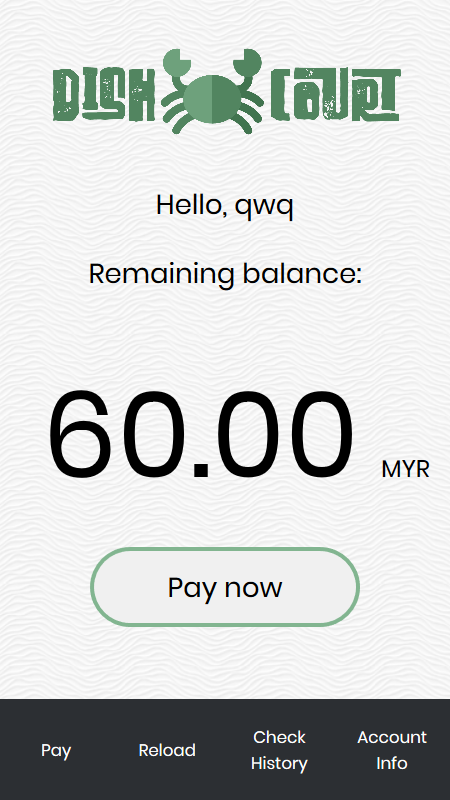


Figure 58

The above figure shows the home page after logging in. The username and remaining balance of the user is shown. The button in the middle now takes the user to the Pay page. The buttons on the navigation bar now redirects the user to their respective pages.

### Pay

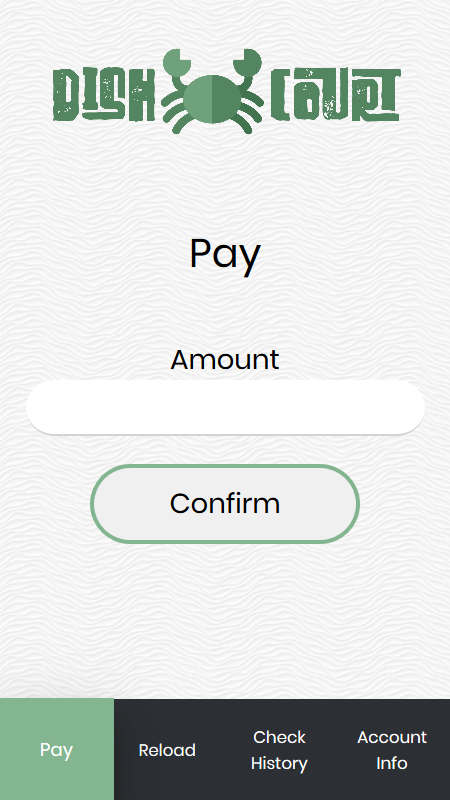


Figure 59

The above figure shows the Pay page. Users use this page to input the amount that they need to pay into the text box and click the Confirm button to generate the QR code to be scanned and charged from the balance.

### Pay with QR code

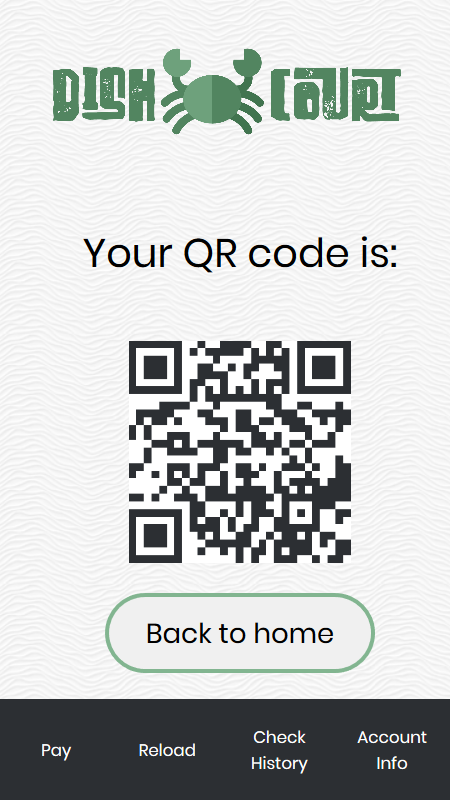


Figure 60

The above figure shows the page where the QR code is generated according to the amount to be paid and the user ID that the amount is to be deducted from. The QR code shown above contains the following string: 2000001MYR12. After the code is scanned by the cashier, the user can click on the Back to home button to go back to the home page.

### Reload

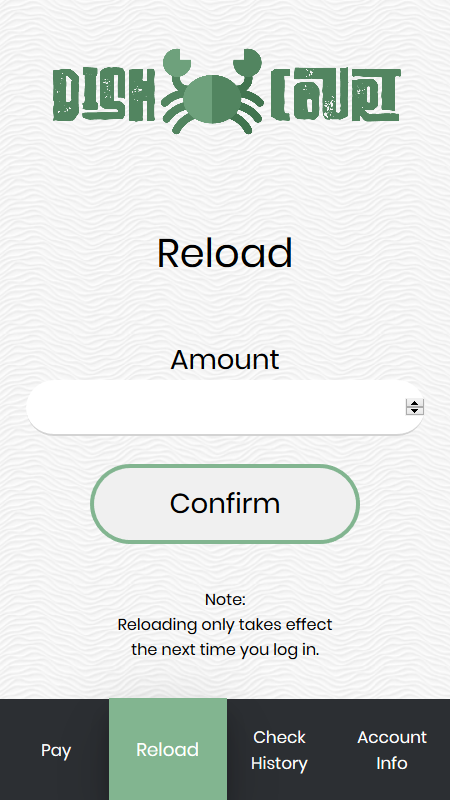


Figure 61

The figure above shows the reload page. Users can reload their account balance by entering the amount that they want to reload into the text box and click Confirm. The button will redirect the user to a page where online banking is used to add balance into the customer account.

### Reload successful

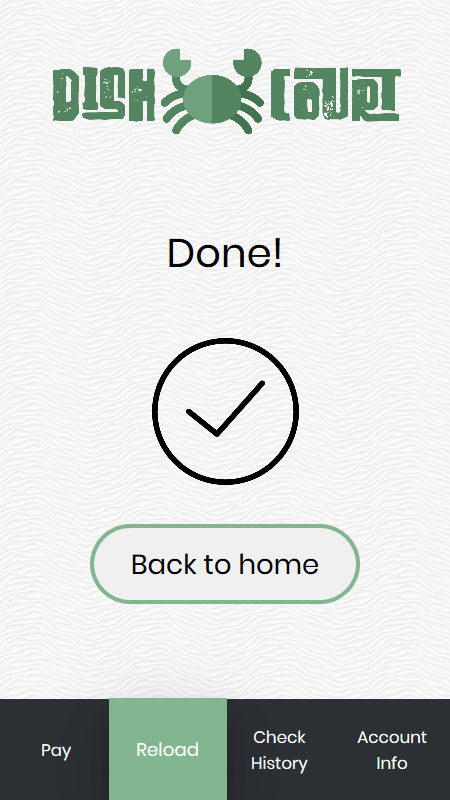


Figure 62

The figure above shows the page that indicates that the reloading process is complete. The button takes the user back to the home page. The remaining balance in the home page should be updated.

### Transaction History

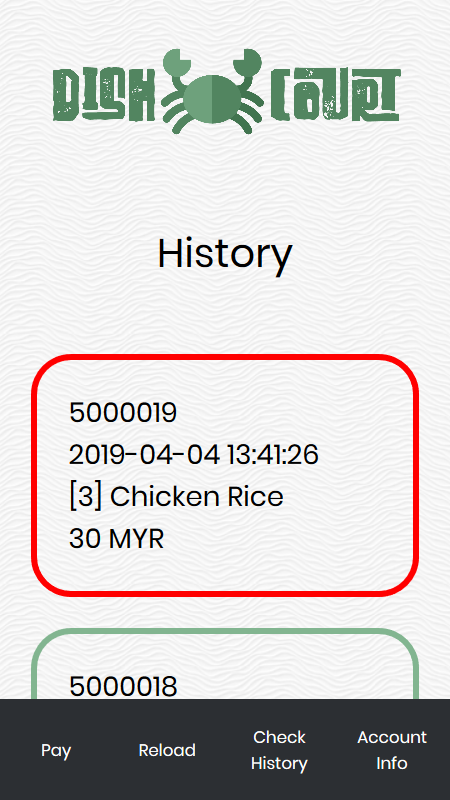


Figure 63

The figure above shows the transaction history page. This page shows a list of orders that the user has performed using the system. If the order was cancelled, the colour of the border will be red in colour. Else, the colour is green. Every instance of an order shows the order ID, date and time of the order, the quantity and item name, and the total price charged for the order.

### Account settings

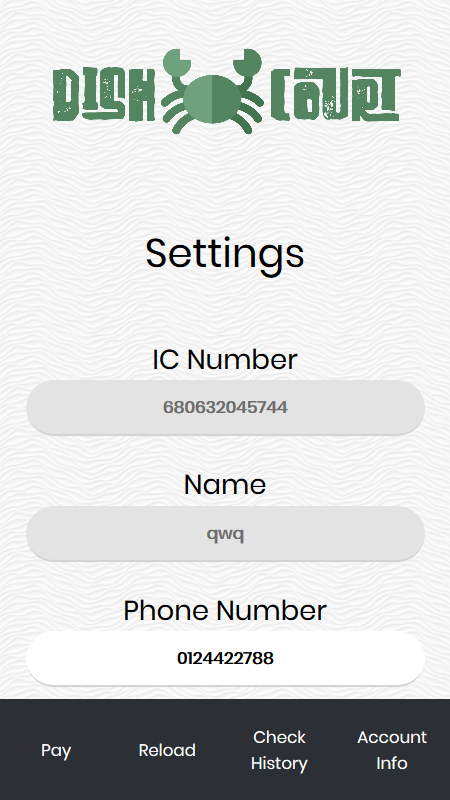


Figure 64

The figure above shows the Account Settings page. This page allows the users to change some details associated with their accounts. The only details that the customers are allowed to modify are phone number, email address, username, and password.

### Register account

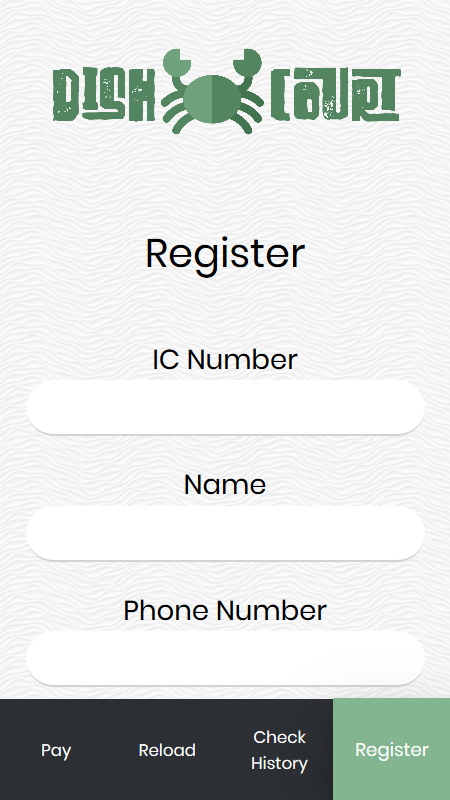


Figure 65

# The above figure shows the Register page. This page is for users that are potential customers to register an account in the system for future use. The user fills in the details and click the Register. If the registration is successful, the user will be redirected to the login page.15.0 Sample Source Codes

In our Part A of the system, we used various languages and library to power the system, namely JQuery, JavaScript, MySQL and PHP. We also used various JavaScript and JQuery-based plug-ins such as “jquery-sumtr (DLarsen, 2013)”, “jQuery Modal (kylefox, 2012)”.

There is also a considerable amount of AJAX methodology used in this system. This is because AJAX can provide a more seamless user experience compared to usual method, which is to refresh the entire page when it needs data. It can also reduce server load by requesting only the required file, not the entire document (Shan, 2013).

Below are a few notable examples of codes that used technologies above:

## 15.1 POS Interface



Figure 66: POS Interface Code

In the code above, it is a snippet that is used to determine whether the code is a item barcode or a payment barcode. If the code is a payment barcode, it’ll be inserted into payment page; if the code is a item barcode, it’ll be used to run a database query to find the item, and add it into table. First, it checks if the code contains the word “myr” as payment barcode is formatted as “-customerid-“MYR”-pay amount-”. If the word MYR is not found in the barcode string, it’ll proceed searching the database with the entered barcode. If the database returns nothing, it’ll alert the user, else, it’ll add the collected data into the table below.

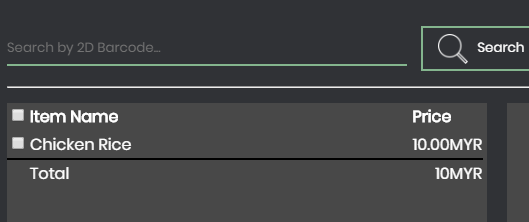


Figure 67: Total price

The figure above shows the data is being added into the table. As you can see, there is a “Total” field below the Price field. This is done by using sumtr plugin by DLarsen (DLarsen, 2013).

$('#itemPOS').sumtr({

readValue:function(e){

return parseFloat(e.html().replace(/[^09\.]+/g,"")); },

formatValue : function(val) { return val + 'MYR'; },

});

To actually sum up the “Price” column, the above snippet is the line of jQuery code. What sumtr does is essentially adds up all the cells with the class “sum” and show the sum in the cell with “summary-total” field. The “summary-total” field is set in the footer so that it’ll always stay at the bottom of the table.

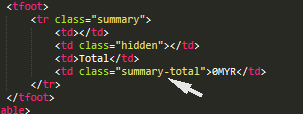


Figure 68: Hidden class

## 15.2 Transaction History

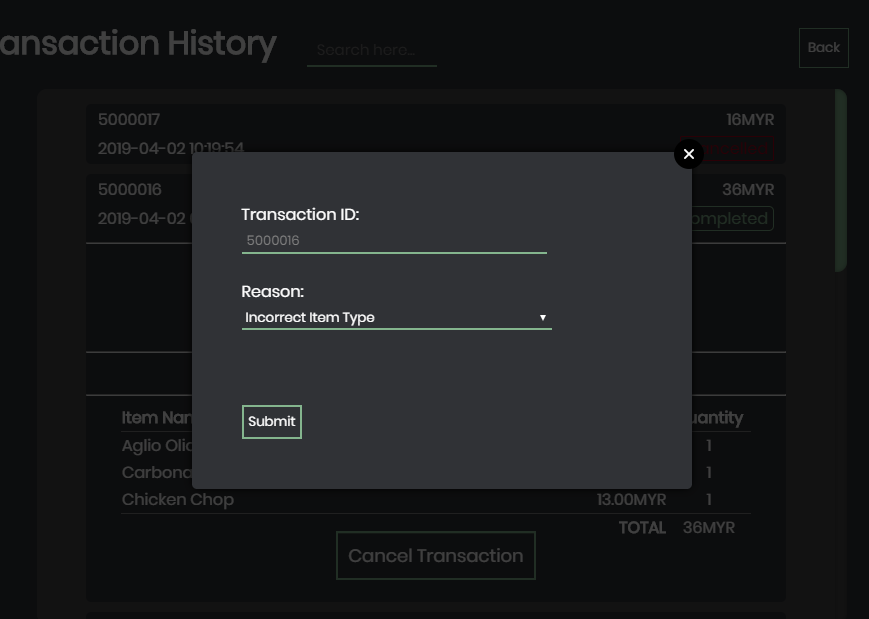


Figure 69: Cancel transaction

In the transaction history page, a modal window is used to allow user to enter reason why are they cancelling the transaction. A modal window is any type of window that is a child (secondary window) to a parent window and usurps the parent's control. A user may not press any controls or enter any information on the parent window until the modal has been closed (Beal, n.d.). To create a modal, a jQuery plug in is used, which is called jQueryModal (kylefox, 2012). It allows a totally different page to be loaded and show on modal, without navigating away the parent window.

$('a.open-modal').click(function(event) {

$(this).modal({

fadeDuration: 250,

fadeDelay: 1.5

});

return false;

});

Snippet above shows how the modal is generated. It simply selects the ‘a’ tag with the class “open-modal” and when user clicks it, it shows the modal.

<a class="a-loginBtn cancelBtn open-modal" href="PHPScript/canceltx.php?txid='.$txid.'&custid='.$custid.'&daterange='.$datediff.'" rel="modal:open">Cancel Transaction</a>

Above show the snippet that triggers the modal. The highlighted part shows that the class “open-modal” is present, and the target page is canceltx.php. It also carries the transaction ID and date difference with the link, to determine which transaction is being cancelled and how old was the transaction.

## 15.3 Generate Detailed Report

This part of the system does not have a lot of fancy stylings, instead, it focuses on what information is being presented instead. Although this is a static website, it still requires a considerable amount of codes to generate.

In the first example, it’ll be the code to find out best selling item of the month. In this case, this SQL query is being used:

SELECT `order`.itemID, item.itemName, sum(itemQuantity) as total from `order` right join item on `order`.itemID = item.itemID where `order`.status='1' and month(datetime) = ".$month." group by itemID order by total desc limit 10

What this snippet essentially does is combine the table `order` and `item`. This is because we have to find the name of the item, which `order` table does not records. Displaying plain item ID, which is stored in the database, is a very tedious task for a user to find out which item ID belongs to what item; thus, we have to find out the item name. This is done by joining two table with their foreign key. It also limits the month to the target month the user wants to generate report of.

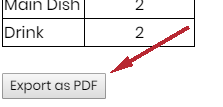


Figure 70: Export report

In our report, there is also a function, which is to export the report as a PDF. To generate a PDF for the page, it involves various modules in Electron . Electron provides 2 IPC (Inter Process Communication modules called ipcMain and ipcRenderer. ipcMain modules is used to communicate asynchronously from the main process to renderer processes, while ipcRenderer is used to communicate asynchronously from a renderer process to the main process. (tutorialspoint, n.d.)

First off, to create the pdf file, we have to add a few things into our main.js file:-

const fs = require('fs');

const os = require('os');

const shell = electron.shell;

const ipc = electron.ipcMain;

The snippet above shows what should be added into the head of main.js.

‘const fs = require(‘fs’)’ is a node.js module that provides an API for interacting with the file system in the computer (Node.js, n.d.).

‘const os = require(‘os’)’ is a module that allows us to create a temporary location to store the pdf file

‘const ipc = electron,ipcMain;’ is an instance of the EventEmitter class. When used in the main process, it handles asynchronous and synchronous messages sent from a renderer process (web page). Messages sent from a renderer will be emitted to this module. (Sathish, 2018)

‘const shell = electron.shell’ is a module that provides functions related to desktop integration, which means it can do some operations like open file, delete file, make shortcut link etc. (Electron, n.d.)

At the end of main.js file, we have to add a bunch of codes to convert the content into PDF file.

ipc.on('print-to-pdf', event => {

const pdfPath = path.join(os.tmpdir(), 'detailedReport.pdf');

const win = BrowserWindow.fromWebContents(event.sender);

win.webContents.printToPDF({}, (error, data) => {

if (error) return console.log(error.message);

fs.writeFile(pdfPath, data, err => {

if (err) return console.log(err.message);

shell.openExternal('file://' + pdfPath);

event.sender.send('wrote-pdf', pdfPath);

})

})

});

What the snippet above essentially does is create variable named pdfPath, which allows the application to store the PDF in a temporary location using the ‘os’ module we imported earlier, and ‘win’ to grab the content of the window.

After that, the method ‘printToPDF’ can be used here to convert the ‘win’ page to PDF (Electron, n.d.). Then, the fs module is used to write the pdf file. After writing the file, the code will send an event called “wrote-pdf” to renderer.js, which, in our case, is included in the detailedReport.php.

const ipc = require('electron').ipcRenderer;

const printPDFButton = document.getElementById('print-pdf');

printPDFButton.addEventListener('click', event => {

ipc.send('print-to-pdf');

});

ipc.on('wrote-pdf', (event, path) => {

const message = `Wrote pdf to : ${path}`;

document.getElementById('pdf-path').innerHTML=message;

})

What the snippet above does is first, making itself a renderer process. After that, a click event is added to the printPDFButton. If the button is clicked, we will send an event called ‘print-to-pdf’ to main.js to handle the conversion process.

After converting the file, the ‘wrote-pdf’ event will be emitted, and the generated pdf file will be opened after it is done.

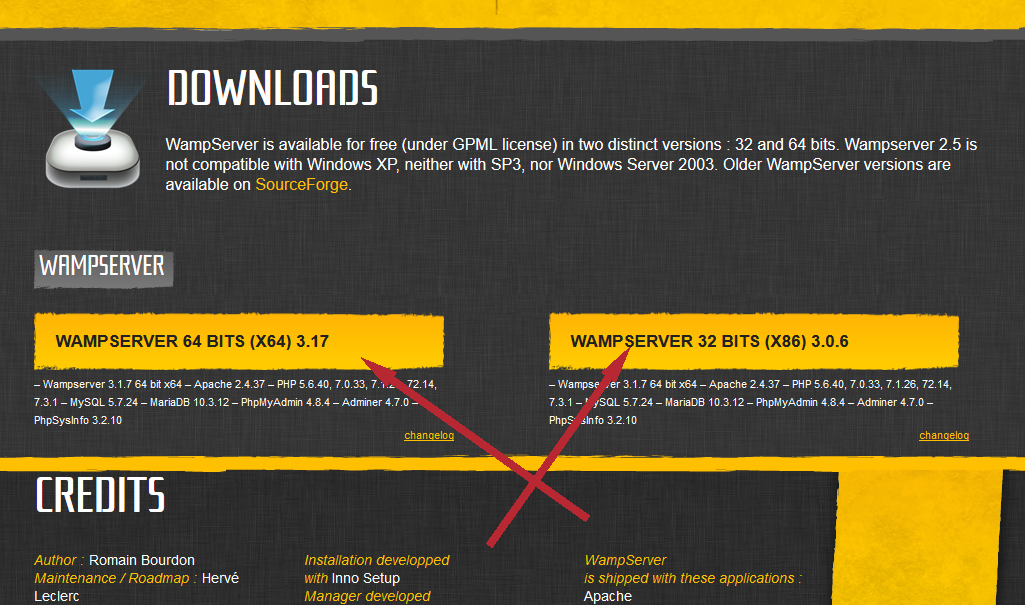
## Installation Guide

In this section, we will be going through step by step tutorial to get this system running. To install this system, it requires a person who is good at computer to deploy this system in one’s computer.

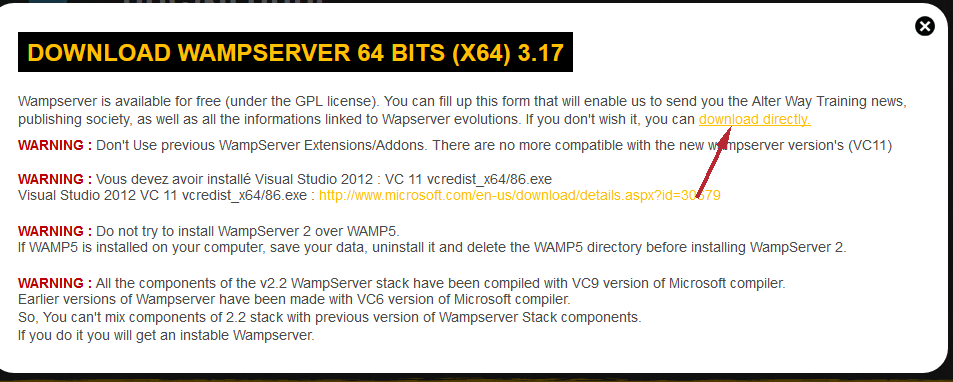
Software Required: WampServer

### Installing WampServer

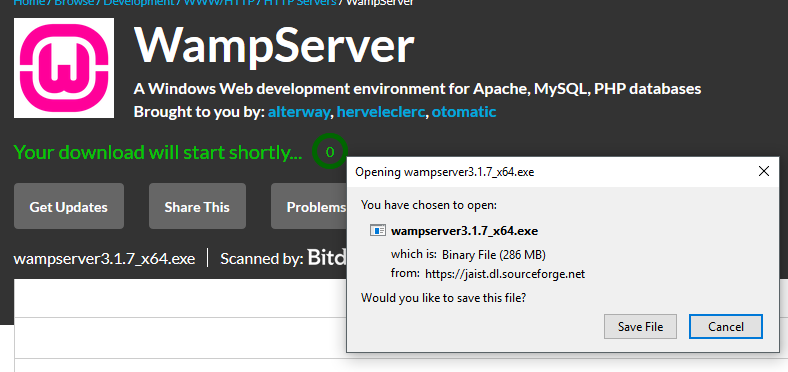
1. Head to WampServer official website: <http://www.wampserver.com/en/> , scroll down to DOWNLOADS section and click on relevant button



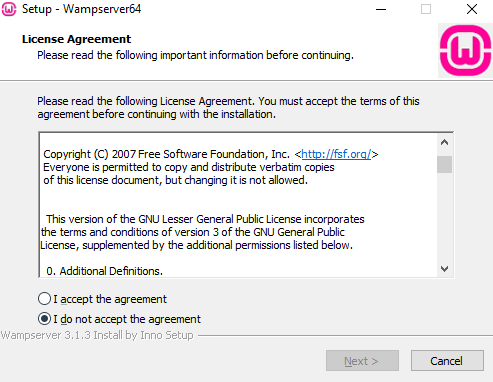
1. Tap “download directly” at the pop-up modal



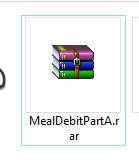
1. You will be redirected to sourceforge.com, wait a few seconds, and the download should be starting.



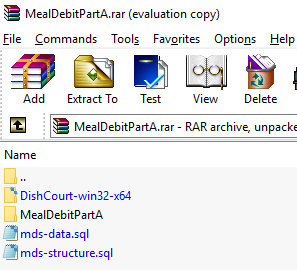
1. Follow the instructions on the screen and WampServer should be installed.



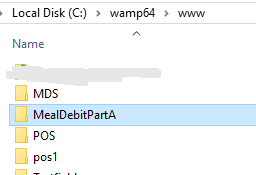
### Installing the system



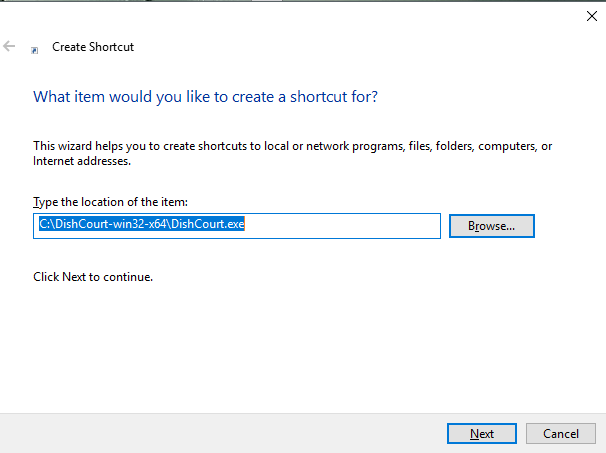
1. All the required files to run the system is packed in this zip. The md5 checksum for this compressed file is 70355FCB1F0FFF9FD94798621B0607E9.



1. This compressed file contains four files: mds-data.sql, mds-structure.sql, DishCourt-win32-x64 and MealDebitPartA. Extract the file.

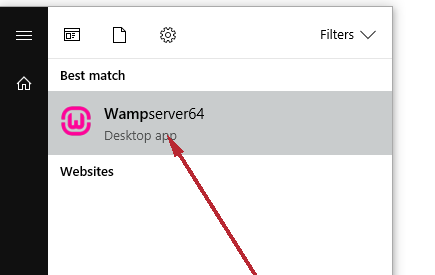


1. Copy “MealDebitPartA” to “your-wamp-directory/wamp64/www” folder.
2. Copy DishCourt-win32-x64 folder to any places in your drive.



1. (Optional) Create a shortcut of DishCourt.exe on desktop for easier access.

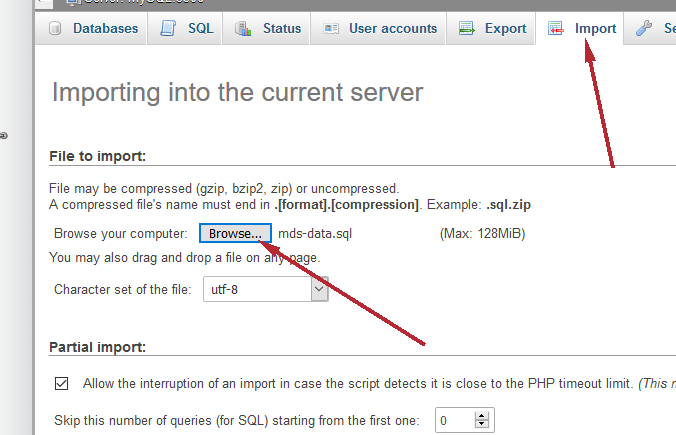
### Setting up database



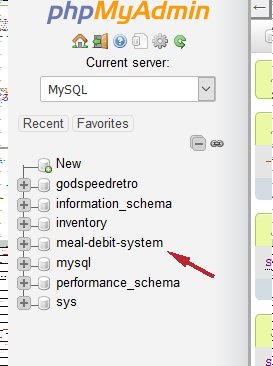
1. Launch WAMPServer64



1. Go to localhost/phpMyAdmin/index.php and log in



1. Once logged in, go to “Import” Tab, and select the .sql file included.
   * If you need a database with sample data, go for mds-data.sql
   * If you only need a database with its structure, import mds-structure.sql



1. Once the database is added to the left panel, the system is ready to be used.
2. Run the application by running DishCourt.exe
3. Log in as admin with ID “1000001” and PIN “123456” to set up the system.