Software Design Specification

Restaurant Ordering System

Prepared by: Abbey Herter, JJ Rezaei, and Anthony Ngo

October 13th, 2023

Table Of Contents:

System Description:

The Restaurant Ordering System is a simple software system that makes the order-taking process easier and more efficient for restaurant employees and management. The primary goal of this system is to ensure customer satisfaction and user experience by maintaining straightforward and quick functionality of the system.

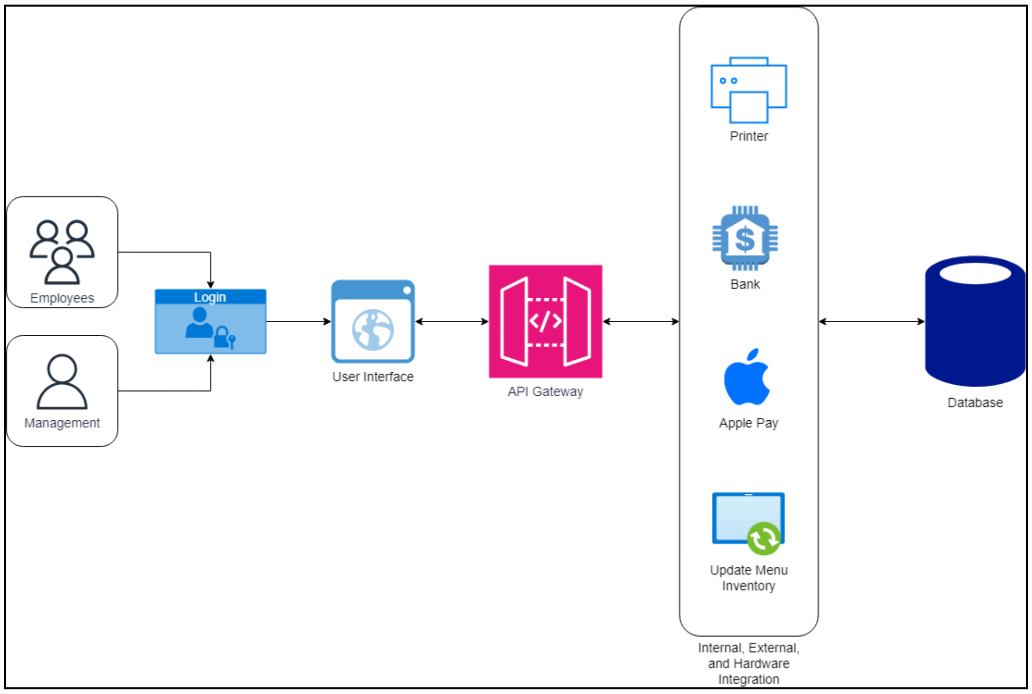
Users, the employees and management, will be able to login to the system with their own special account, access the user interface where they will place a customer's order, complete customer payment, and output a receipt for both the customer and the kitchen. Management will have additional control that will allow them to update the inventory and access database information as well.

Keeping a user-friendly system will not only benefit the restaurant with the ability to produce more orders but also guarantees a quicker runtime for the system as well.

This Software Design Specification includes Software Architecture Diagrams and UML Class Diagrams, both provided with visuals and descriptions to enhance the comprehension of the system's functions including its components, classes, attributes, and operations.

Software Architecture Overview:

Restaurant Ordering System Software Architecture Diagram



Software Architecture Diagram Descriptions:

Management: This is a user or users of the system that have administrative and authoritative functions over the system. This user will have to surpass a login page before accessing the UI. Users have the ability to oversee and control parts of the software system including updating the menu inventory along with backing up the inventory so that the information in the UI is up to date.

Employees: This is a user or users of the system whose role is to successfully navigate through the UI so that the order information is sent through the API Gateway and to the Internal, External, and Hardware Integration such as the printer, bank, or 3rd party systems. Users will have to surpass a login before accessing the UI as well.

Login: The login page acts as a security gateway to the UI. The page prompts the user to input their login information in order to authenticate their access.

User Interface (UI): The User Interface (UI) is a simple interface that allows the user to interact with different features and navigate through the system efficiently. These features include selecting if the order is dined in or if the order is to-go along with creating the order for the customer. Users will also be prompted to administer the customer's payment which will be sent from the UI through the API Gateway to the bank or Apple Pay. Furthermore, users will then be prompted to print a receipt which will be sent from the UI to go through the API Gateway to then reach the printer.

API Gateway: The API Gateway acts as the middleman in this Software Architecture Diagram and manages communication between the UI and the Internal, External, and Hardware Integration. The API Gateways functions include gathering, transforming, and requesting data along with routing components together.

Internal, External, and Hardware Integration: These are the components that communicate through the API Gateway back to the UI to complete requests made by the user.

Printer: The printer is a hardware component that is prompted by the user to output the customer's receipt and the receipt for the kitchen.

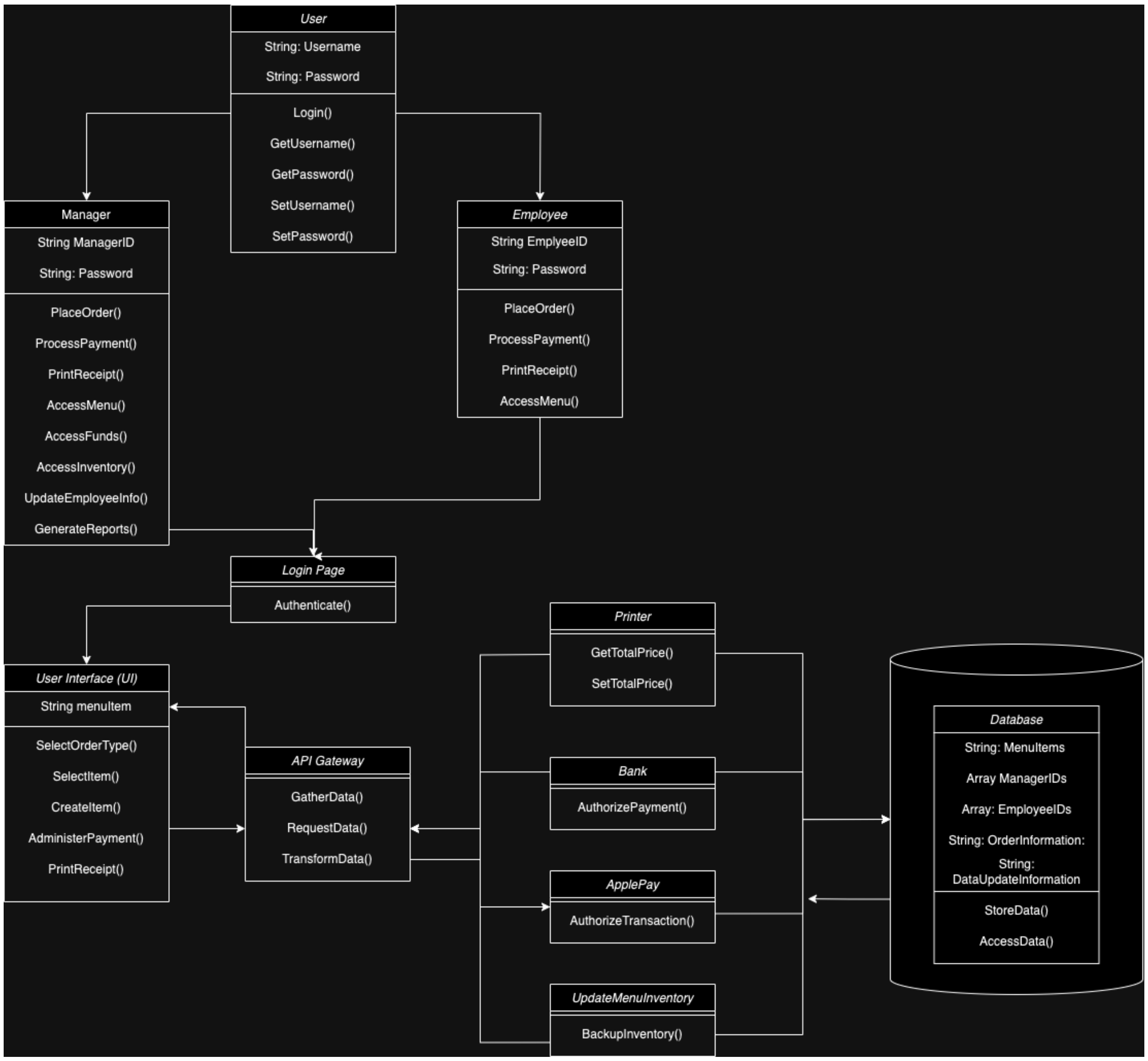
Bank: The bank is an external service that will authorize the payment given by the customer.

Apple Pay: This is a third-party payment service that will authorize the transaction given by the customer through insertion or tap.

Update Menu Inventory: This is an internal integration that allows management access to frequently backup and update the menu inventory including changing pricing, stock amounts, menu additions, and menu removals. This integration will check those features and communicate the feedback through the API Gateway to the UI.

Database: The database is an important internal component of the system that contains all of the data storage. Within the database is the menu item names, prices, and descriptions along with order information and data update information. All of the information that has traveled from the UI through the API Gateway to the Internal, External, and Hardware Interfaces is then stored in the database and is able to be accessed by the management.

UML Class Diagram:



UML Class Diagram Descriptions:

User:

EmployeeID string - used to log into the employee account

Password string - used to log into employee account

Login() void - logs employee to either manager or employee account depending on employee ID or manager ID

getUsername() string - retrieves employee username from database

getPassword() string - retrieves password of login from database

setUsername(string) string - sets username information in database to variable

setPassword(string) string - sets password information in database to variable

Manager:

ManagerID string - used to log into the manager account

Password string - used to log into manager account

PlaceOrder() void - places an order for a customer

ProcessPayment() void - processes the payment for the customer

PrintReceipt() void - prints the receipt for the customer

AccessMenu() void - returns the menu contents (stored in the database)

AccessFunds() void - returns the funds available for the restaurant (stored in the database)

UpdateEmployeeInfo() void - used to modify or delete employee information

Employee:

PlaceOrder() void - places an order for a customer

ProcessPayment() void - processes the payment for the customer

PrintReceipt() void - prints the receipt for the customer

AccessMenu() string - returns the menuItems

Login Page:

Authenticate() bool - ensures that the previous committed action attempted is valid

User Interface (UI):

SelectOrderType() void - sets a certain order type to item (to-go or dine-in)

SelectItem() void - selects a menu Item and adds it to the order

CreateItem() void - creates a new menu item

AdministerPayment() void - administers a type of payment for the customer

PrintReceipt() void - sends a print request to the printer

API Gateway:

GatherData() void - aggregates the data from database into array of strings

RequestData() string[] - returns data aggregation from gatherData

TransformData() void - allows user to change data from database

Printer:

GetTotalPrice() int - gets and calculates price of items selected

SetTotalPrice(int) int - sets the total price of the items selected to a variable

Bank:

AuthorizePayment() void - ensures payment is valid

ApplePay:

AuthorizeTransaction() void - ensures payment is valid

UpdateMenuInventory:

BackupInventory() void - creates a backup inventory log with all database info

Database:

StoreData() void - stores data into the database

AccessData() string - returns data from the database

Development Plan and Timeline

Partitioning of Tasks:

We decided to partition the tasks as equally as possible. Abbey handled the software architecture overview and its description. Anthony handled the UML diagram with some input from JJ and JJ handled the UML class diagram descriptions. We thought this would all take approximately the same amount of time for each individual, as each section would take the individuals an estimated 1-2 hours to complete.

Team Member Responsibilities:

As mentioned previously, Abbey handled the software architecture overview and description. This means ensuring that the software architecture is accurate to the group's vision and understanding of the program. Anthony drew out the UML diagram, meaning he needed to focus on what the implementation of the software architecture diagram and its overview would look like. Finally, JJ handled the UML diagram descriptions. This meant specifying the UML diagram to be as clear as possible how the program will finally be implemented.

Test Plan

Updated UML Diagram:

After reviewing our most recent submission we have decided to update our SWA and UML diagrams. We noticed that in the SWA Diagram and the UML Diagrams, the arrows needed to be updated to a double-sided arrow when going from the UI, the API Gateway, the Internal, External, and Hardware Integration, and the Database. Additionally, we modified the UML Diagram and we changed selectItemType to selectItem and createItemType to createItem. We also added an attribute to the UI class; a string menuItem.

Test A: Login

<u>Unit Test A:</u>
Verify if a user will be able to log in with a valid username and password.

```
if (username valid && password valid)
    Give permissions based on ranking (manager or employee)
else
```

```
        Do nothing
```

This unit test is an if-else statement that checks if the username and the password are valid, if both are true then the user will be able to log in depending on their employee status, if the username and password are false then the else statement will execute which is to do nothing. This unit test uses the User class.

<u>Integration Test A</u>
Manager requests Employee ID.

```
Employee employee = new Employee(#);

EmployeeID = employee.getEmployeeID()

if (EmployeeID is null)
    return employee not exist
else
    return EmployeeID
```

This integration test is an if-else statement that checks if the EmployeeID is null if it is true then it returns that the employee's ID does not exist, if the statement is false then the employee's ID exists and returns the ID. This integration test uses multiple parts of the UML Diagram classes including the Employee and the Database.

<u>System Test A</u>
Manager logs in and then creates an Employee ID

```
if (ManagerID && Password)
    Random generate EmployeeID
    if (EmployeeID does not exist)
        StoreData(EmployeeID)
    else
        Return to random generate EmployeeID
else
    Reenter ManagerID and Password (set amount of tries)
```

This system test is a double if-else statement, first it checks if the ManagerID and password are true which allows the Manager to login, then the manager will receive a random generated EmployeeID the next if-else statement will check if the EmployeeID does not exist then it will store that EmployeeID in the Database, if the EmployeeID exists then the manager will be prompted to return to the random generator and do this until an EmployeeID that does not exist is found. Then the statement exits and if the ManagerID and password are false then the manager is promoted to reinput the ManagerID and password a set amount of tries to log in. This system test uses the complete system of the UML Diagram classes including the User, Manager, Login Page, and Database.

Test B: Placing An Order

<u>Unit Test B</u>
Create a new item on the menu

```
menu = Menu()  # menu class (constructing the menu)
menuItem = menuGetItem“”

if (menuItem is null)
        CreateItem(menuItem)
else
        Do nothing

if (menuItem) is not none:
        print “item created”
else:
        print “item not created”
```

This unit test accesses the menu class to see if the menu item already exists if it doesn't, create it (assuming we have a createItem() function). It then checks to see if an item is created and prints out whether or not it was created.

<u>Integration Test B</u>
Accessing order information from database

```
database = database() #accessing database class
orderID = 100

orderInfo = database.getOrderInfo(orderID)

if orderInfo is not none:
        print “order info grabbed”
else:
        print “order info not grabbed”
```

This integration test accesses the database class (with the assumption that we have a database class). We then utilize a getter to get the OrderInfo and follow it up with an if/else statement to verify if the orderInfo was retrieved.

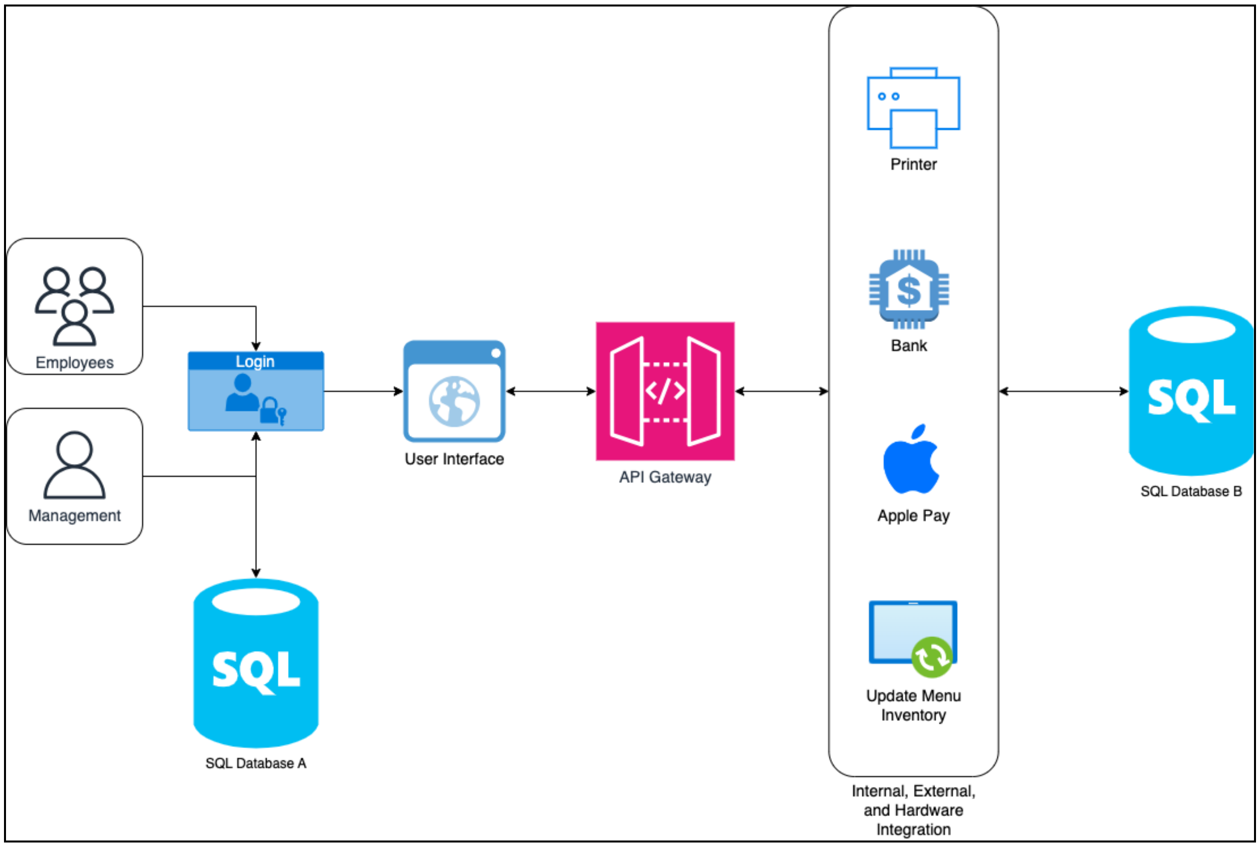
System Test B
User creates order to check if available administer payment print receipt

```
# creates order
createOrder(orderID, items)

administerPayment(order, PaymentInfo)

if not orderPaymentProcessed:
    processPayment()
    printReceipt()
else:
    Print"payment has already been processed successfully"
    printReceipt()
```

This system test creates an order using a createOrder function with the parameters orderID, and items. Once an order is created it attempts to administer payment and checks using an if else statement. Within the statements, it finally attempts to print a receipt.

Software Design 2.0:

Software Architecture Diagram Update:



Our original Software Architecture Diagram, shown on page 3, had a database but it was not specified. For this Software Design 2.0 we have decided that our system needs two SQL Databases, A and B, to represent the management and employee login data and the restaurant menu and order information data.

Data Management Tradeoff Strategy:

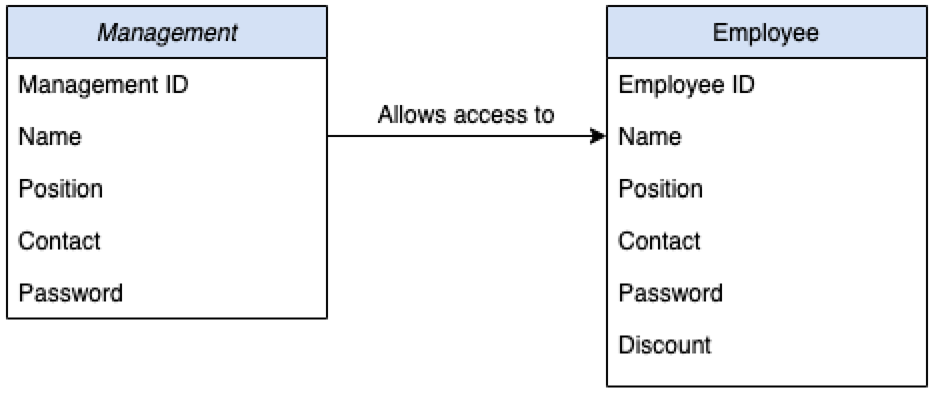
> <u>SQL:</u> For our Data Management we have decided to use SQL. We decided that we would need two SQL databases because it is more efficient to maintain the login information of the management and employees into the first database A and store the menu inventory and customer orders within the second database B. We chose SQL because it is scalable which will allow the developers to scale each record independently without the risk of overlap from another database. Additionally, SQL will promote our performance by optimizing the fields for the databases for example having user authentication as a login field, and specific menu ID's as an inventory field.

> <u>Possible Alternatives Solutions:</u>

Another solution that would be less complex and consistent then SQL is NoSQL. NoSQL is more flexible than SQL, however, because we believe it is better to have multiple databases we are willing to give this up since our data is fairly structured so there isn't much need for a lot of flexibility.

Data Management Diagram & Descriptions:

SQL Database A:



Management
- Management ID: (int) Provided ID number that the management is assigned with.
- Name: (string) The name of the management user.
- Position: (string) The job title the management user is.
- Contact: (string) The phone number or email address of the management.
- Password: (string) The management password required for accessing the UI.

Management Example:

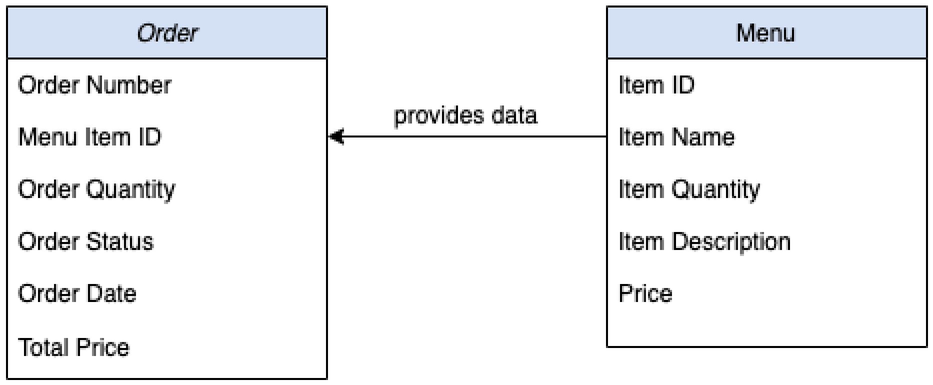
Management ID	**Name**	**Position**	**Contact**	**Password**
16790822	John Doe	Chef	800-100-1090	flower123!

Employee
- Employee ID: (int) Provided ID number that the employee is assigned with.
- Name: (string) The name of the employee.
- Position: (string) The job title of the employee.
- Contact: (string) The phone number or email address of the employee.
- Password: (string) The employee's password required for accessing the UI.
- Discount: (int) The discount amount applied to their account when ordering.

Employee Example:

Employee ID	**Name**	**Position**	**Contact**	**Password**	**Discount**
18844671	Jane Doe	Cashier	801-101-1091	happy678?	20%

<u>SQL Database B:</u>



Menu

- <u>Item ID:</u> (int) A number to represent what the item type is.
- <u>Item Name:</u> (string) The name of the inventory item.
- <u>Item Quantity:</u> (int) The amount of the item in stock.
- <u>Item Description:</u> (string) A short description of what the item is.
- <u>Price:</u> (double) The price of the inventory item.

Menu Example:

Item ID	**Item Name**	**Item Quantity**	**Item Description**	**Price**
17	Coca Cola	300	Drink/Soda beverage	2.50

Order

- <u>Order Number:</u> (int) A special number to indicate the customer's order.
- <u>Menu Item ID</u>: (int) A number to find the item the customer ordered.
- <u>Order Quantity:</u> (int) The amount ordered of that specified menu item.

- Order Status: (string) Tells the user if the order is complete, being processed, or incomplete.
- Order Date: (string) The date the order was placed.
- Total Price: (double) The calculated price the customer will pay for the menu item and the quantity of that item ordered.

Order Example:

Order Number	Menu Item ID	Order Quantity	Order Status	Order Date	Total Price
44558901	17	2	Complete	11-09-2023	5.00