J’s Restaurant Automation

[SWE 3313 Section 01] | [Professor Sekar]

Team 7

Ian Riggins, James Ross, John Sheffield, Jordan Rainford

2023

**Table of Contents**

[**Project Plan Documents** 2](#_Toc133783764)

[Scope 3](#_Toc133783765)

[Schedule + Gantt Chart 4](#_Toc133783766)

[Team Organization 5](#_Toc133783767)

[**Data Management Plan + Technical Description** 6](#_Toc133783768)

[**Resumes’** 7](#_Toc133783769)

[**Requirements Documents** 12](#_Toc133783770)

[Test Plan 13](#_Toc133783771)

[Functional Requirements 14](#_Toc133783772)

[Non-Functional Requirements 15](#_Toc133783773)

[Use Case Diagram 16](#_Toc133783774)

[Class Diagrams 17](#_Toc133783775)

[ER Diagram 18](#_Toc133783776)

[State Transition Diagram 19](#_Toc133783777)

[Class Documentation 20](#_Toc133783778)

[**System Design Documents** 22](#_Toc133783779)

[Report formats 23](#_Toc133783780)

[Screen layouts/Screenshots 24](#_Toc133783781)

[The Timeclock Screen 25](#_Toc133783782)

[The Menu Screen 26](#_Toc133783783)

[The Table View Screen 27](#_Toc133783784)

[The Orders Screen 28](#_Toc133783785)

[The Employees Screen 29](#_Toc133783786)

[Full UML Diagram 30](#_Toc133783787)

[Supporting Text Specification 31](#_Toc133783788)

[Technical Support Specification 41](#_Toc133783789)

[Appendix 42](#_Toc133783790)

# **Project Plan Documents**

# 

## Scope

Project Goal: Develop a software that will help automate the restaurant process to make customers time at the restaurant easier and make the restaurant easier to run and operate.

Requirements:

* Employee schedules
* Employee roles
* Employee hours
* Employees pay (before tax)
* Employees pay (after tax)
* Customer order history
* Customer information
* Customer payment method
* Menu items
* Menu ingredients
* Back of house inventory
* Stock management system
* Menu management system
* Billing management system
* Hours of operation
* Popular times
* Total gross revenue
* Total net profit (before employee pay)
* Total net profit (after employee pay)

Goals:

1. Establish operation hours and peak traffic times for the Business
2. Develop *Adaptable* Customer Record
3. Develop *Adaptable* Employee Record
4. *Add* a User-Input feature to the Customer and Employee Record
5. Develop Inventory Management System with User-Input
6. Develop *Adaptable* Database of Menu Items
7. Develop Billing system which integrates with the Menu and Customer Record
8. Create Income-And-Expenses algorithm which can calculate the gross revenue, and net profit both before and after employees are paid

## Schedule + Gantt Chart

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Task ID | WBS | Planned Start | Planned Finish | Anticipated Workload | Actual Workload | Percent Complete |
| 1.1.1 | Customer Record | 2/21/23 | 3/22/23 | 3 | 3 | 100% |
| 1.1.2 | Table Class | 2/21/23 | 3/22/23 | 4 | 4 | 100% |
| 1.2.1 | Employee Record | 2/21/23 | 3/22/23 | 4 | 4 | 100% |
| 1.2.2 | Employee Class | 2/21/23 | 3/22/23 | 3 | 4 | 100% |
| 1.2.3 | Menu/Order Record | 2/21/23 | 3/22/23 | 6 | 7 | 100% |
| 1.2.4 | Item Class | 2/21/23 | 3/22/23 | 7 | 8 | 100% |
| 2.1 | User input Integration | 2/21/23 | 3/22/23 | 5 | 7 | 100% |
| 2.2 | Exception Handling | 2/21/23 | 3/22/23 | 2 | 3 | 100% |
| 3 | Combine Classes | 3/22/23 | 3/23/23 | 5 | 6 | 100% |
| 4 | Billing Algorithm | 3/22/23 | 4/1/23 | 6 | 5 | 100% |
| 5.1 | GUI Creation | 4/1/23 | 4/16/23 | 8 | 6 | 100% |
| 5.2 | GUI Integration | 4/16/23 | 4/24/23 | 8 | 4 | 88% |
| 6 | Testing | 4/24/23 | 4/27/23 | 5 | 4 | 76% |

## 

## Team Organization

Each member of our team will handle unique key elements of writing this program. Jordan Rainford will be the project manager, overseeing all aspects of development. I will take on the responsibility of keeping the project on task and on time, while also placing my efforts in programming wherever they are needed.

Ian will be the Development Leader, which makes him responsible for the specifics of how each piece of code is to be written, how each program going to interact with each other, and a variety of other details of the project.

James will be the lead Software Designer, meaning that James will work closely with Ian to ensure that code concepts are being properly implemented. James will be responsible for the oversight of algorithmic data, as well as the logging of any errors that arise during development.

John will be the Exception Handler, which makes him responsible for catching any possible Exceptions or errors that any algorithm written by the preceding team members can throw. This also means that he must report back to proceeding team members with his findings if an error is too grave for him to correct.

Jordan Rainford – Project Manager / Programmer

Ian Riggins – Development Leader / Integration Specialist

James Ross – Software Design Leader / Programmer

John Sheffield – Exception Handler / Lead Tester

# **Data Management Plan + Technical Description**

Account Management

* Allow users to enter in a username and password to login to an account.
  + If user does not have an account allow a manager to create one.
  + Some accounts will have different permissions from others.
* Each account has a special first and last name that employee or manager are required to fill out.
  + Also requires manager to create or change designated role(s).
  + Option for employee to change their designated name or password.

Menu

* Option that will bring up a digital menu to look at showing food items and their price.
* Does not require you to sign in to look at.
* Manager can change the menu or the prices when signed in.

PlaceOrder

* Allows user to select an item from the menu and add it to their order.
  + option to add special requests to the item.
* Can pay once they added all desired items.

PaymentMethod

* After clicking pay user is brought to a payment screen
  + Can select one of three payment types.
    - Cash
    - Check
    - Credit Card
      * Takes you to the Credit Card center.

Schedule

* Button all employees have access to which displays the current weeks schedule.
* Managers will have option to alter the schedule.

Table View

* Shows all the current tables and their status.
  + If table is red, it is occupied.
  + If table is orange, it is not occupied but has not been bussed.
  + If table is Green, table is unoccupied and it ready to be used.
* Each table has a unique ID that is shown.
* Host, Waiters, bussers, and Managers can view the table layout.

View Queue

* Shows all the orders that are in the Queue and all the relevant information from said order.
* Can be seen by Waiters, Cooks, Managers.
* Queue can be altered by a manager but no one else.

# **Resumes’**



Text

Description automatically generated

Text, letter, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

# **Requirements Documents**

## Test Plan

To ensure we can guarantee that the software we are developing meets all the desired requirements we are going to test our software using two different methods to ensure that every part of the program is working as intended. The two methods which we are going to use to test the software is both white box testing and black box testing. White box testing is where the testing of the software will be done by someone who knows how the software works internally, so they would know what would or would not cause error. On the other hand, black box testing is done by someone who does not know the internal mechanisms of the software and just knows what is being presented to them.

The first method we are going to use to test the software is the white box testing method, this is because we must start by ensuring the basic internal workings of the program are working as intended before we can start working on the external. Because of this reason we are going to test the internal side of the program before working on fine tuning the external side of the program. This is important since we need to ensure that all the internal parts of the software are working as intended and does not have any major oversights. For instance, a reason we would want someone who knows the internal working of the software instead of someone who does not to test the program is because they would know what is likely to cause errors so they would know what to try to ensure no errors do occur.

The second method of testing we are going to do is the opposite of white box testing which is black box testing. Unlike in white box testing where the person doing the testing has knowledge over how the program works internally, in black box testing the person who is testing the program has no knowledge on how the program works internally. The information we gather from this test is going to be extremely important since we are developing this software to be used by people who do not have any knowledge of software engineering. Because of this the person who will be doing the black box testing is similar to the people who are going to be using the software once it has been launched. Because of this reason their opinion of the software is very important and crucial to refining the software to be the highest quality it can be for the client. We will use their opinion on the UI and UX to change parts that should be altered to better help the client use the software while also ensuring that the part of the UI and UX the tester liked stayed the way they were.

## Functional Requirements

* Employee schedules
  + Display the employee’s schedule so employees can view when they work and what they are doing on that day.
* Employee roles
  + Allow employees to login in, after employee is login they will have access to various features depending on what role they are login as. Some people will have access to various different roles they can choose from.
* Employee hours
  + Keep track of how many hours a employee has worked based off of when they clocked in and clocked out.
* Employees pay
  + Display an employee’s current pay before taxes based off of their current hours worked times their hourly pay. Also shows an employee's pay after taxes by subtracting the taxes from the current pay.
* Customer order history
  + History that keeps track of how many of each meal item is ordered and the revenue that each meal item produces.
* Customer payment method
  + Allow the customer to pay in multiple methods, including cash, card, or check.
* Menu items
  + Classes that keep track of all the items on the menu and their price.
* Menu management system
  + When a manager is signed in allow them to change the menu and the prices of the items.
* Hours of operation
  + Display the hours of operation, allow manager to change hours of operation.
* Popular times
  + Keeps tracks of what hours the restaurant is popular and act accordingly.
* Total gross revenue
  + Display the current revenue that the restaurant has generated that day. Also shows the previous day’s revenue.
* Tables
  + Shows a view of the current tables and allows bussers, host, and servers to change the status of the tables.

## Non-Functional Requirements

* Ease of Use
  + The program is intended to be used by people who are knowledgeable on technology, so the program should be easy to use so people can quickly and easily learn to use it
* User friendly design
  + The design should be user friendly so that people can easy figure out what each function does.
* Pleasant aesthetic
  + The software should look pleasant to look at.
* Performance
  + The software should work as quickly as possible as to ensure to save as much employee time as you can, should also be able to run on lower end machines.

## Use Case Diagram

Diagram

Description automatically generated

## Class Diagrams

Graphical user interface, text, application

Description automatically generatedGraphical user interface

Description automatically generated

Diagram

Description automatically generated

## ER Diagram

Diagram

Description automatically generated

State Transition DiagramDiagram

Description automatically generatedDiagram

Description automatically generated

## Class Documentation

Item Class:

* Description: A base class that will be used to create several sub classes. Contains several attributes that the child classes will need such as name, description, and cost. Also include basic functions such as getters and setters. Also has an instance of the Item class in the Item class for additional requests such as adding a de or topping.
* Child Classes include:
  + Appetizer Class: A child class of the Item Class that simply calls the base constructor with the attributes it has been given.
  + Drink Class: A class that calls the base constructor with the name it was given and 2 as the price.
  + Entrée Class: Similar to the Appetizer Class except it also has 2 side attributes with their own functions in the class that control them.
  + Salad Class: A class that is a child of the Item class. It has its own attribute called the AdditionalCost that may or may not be null. It has an overloaded constructor one with just the name and cost. If that constructor is called just called the base constructor. And an overloaded constructor that will do everything the first constructer did but also sets the AdditionalCost variable to the additional float it was given.
  + Sandwich Class: Very similar to the Entrée class except it only has 1 side as opposed to 2. It used for wraps, burgers, and Sandwich’s not just Sandwich’s.

Order Class:

* Description: A class that consists of an arrayList of instances of the item class. It also keeps track of when the order was placed using the C# DateTime variable type and a variable to keep track of the total cost of the order. Has 2 constructors one of them creates an empty order and does not take in any variables, the other creates an instance of the order class with a arrayList that was given though an attribute. Contains a function that iterates though the entire arrayList and calculates the totalcost. Also has a function to add and remove an item from the order and calculate the new total cost.

Table Class:

* Description: Class that has several variables including a static variable to keep track of and create the **TableID**. Also has a integer variable to keep track of the total amount of **patron** that has been seated at the table. Also has a byte variable that keeps track of the tables **current** **states**, state 1 indicates table is open, state 2 indicates table is unoccupied but dirty, state 3 indicates table is occupied, state 4 indicates table is unavailable. Also has a variable that includes a **DateTime** variable to keep track of when the current customer was seated. Has a constructor that sets the variables, has a function to seat a customer and a function to be called when a patron leave

Menu Class:

* A class that is initialized at the start of the program being ran which includes several lists for each Item on the menu. There is a list for Appetizers, Salads, Entrée, Sandwiches, Wraps, Burgers, Drink, Sides. Each list contains the item available on the menu and it’s cost. It gets the available items and cost from a .txt document which contains the Item name and price and occasionally additional data. The menu class has a constructor that takes in a string, said string is the file path to the .txt file that will be used by the constructor to initialize menu class. Have a method to print the individual Items contained in a list called PrintList(). Also has a method called PrintMenu() that calls the PrintList() method for every list contained in the Menu.

Employee Class:

* A class designed to keep track of information about the employee who is currently using the software. Will consist of information about the employee such as the employee’s first, middle, and last name, as well as their gender. It also keeps track of an employee's EmployeeNumber used to identify them. It also keeps track of whether or not an employee has restricted access, and what type of employee they are. Has a constructer that sets the first middle and last name along with the ID from given attributes. Also has an overloaded constructor that takes in a no attributes and just sets the attributes to a default value. Has variables getters and setter for each attribute of the class.
* Child Classes include:
  + Manager: A child class that takes in several attributes and calls the base constructor with given attributes, and calls the Unrestrict function from the Employee class, and sets a new double variable called Salary to a given attribute. Also has a getter and setter for the Salary attribute.

# **System Design Documents**

## Report formats

The application will provide a variety of different report formats to different parties, including:

1. Register Receipt + Order Slip
2. Punch-In Screen
3. Punch-Out Screen
4. Monthly Sales Report

## Screen layouts/Screenshots

The application will consist of a wide variety of screens, which are, again, functionally dependent on who is utilizing the application. These screens include:

The Home Screen

Graphical user interface, application

Description automatically generated

Diagram

Description automatically generatedDiagram

Description automatically generatedAccessible to the Manager, Host/Hostess, Waiter/Waitress, Busboy, and Cook. This screen will allow the user to access the key features and other screens of this application. This screen will choose exactly which screens the user will be presented with depending on the Employee Type of the user (Manager, Busboy, Cook, etc.). Below, you will see some variations of the home screen, as well as what employee type they correlate with.

*Figure a: Cook Home-screen Figure b: Manager Home-screen*

## The Timeclock Screen

Graphical user interface, application

Description automatically generated

*Graphical user interface, text, chat or text message, website

Description automatically generated*Graphical user interface, application, website

Description automatically generatedAccessible to the Manager, Host/Hostess, Waiter/Waitress, Busboy, and Cook. This screen will allow the user to clock in/out of the system. The is screen will require a valid EmployeeID to be entered. If the user incorrectly enters their ID, or enters a fake ID, then an error code will prompt them to try again. This will repeat in a cyclical manner until a correct ID is given, thus allowing the user to clock in/out. Below, you will see variations of the Clock-In/out screen as you key in your Employee ID.

*Figure c: Clock-In Prompt Figure d: Clock-Out Screen*

## The Menu Screen

Graphical user interface, text, application, chat or text message

Description automatically generated

Graphical user interface, application, chat or text message

Description automatically generatedAccessible to the Manager and the Waiter/Waitress. This screen will let the user submit an order. This screen displays all variations for any menu item, as well as the prices, descriptions, and add-ons. This screen requires that at least one item be selected, and that the user does not exceed the limitations for certain choices (i.e., no more than two sides to an entrée). Below, you will see various sub-levels of the menu screen, as well as a few of the error messages which ensure all requirements are met.

Graphical user interface, text, application, chat or text message

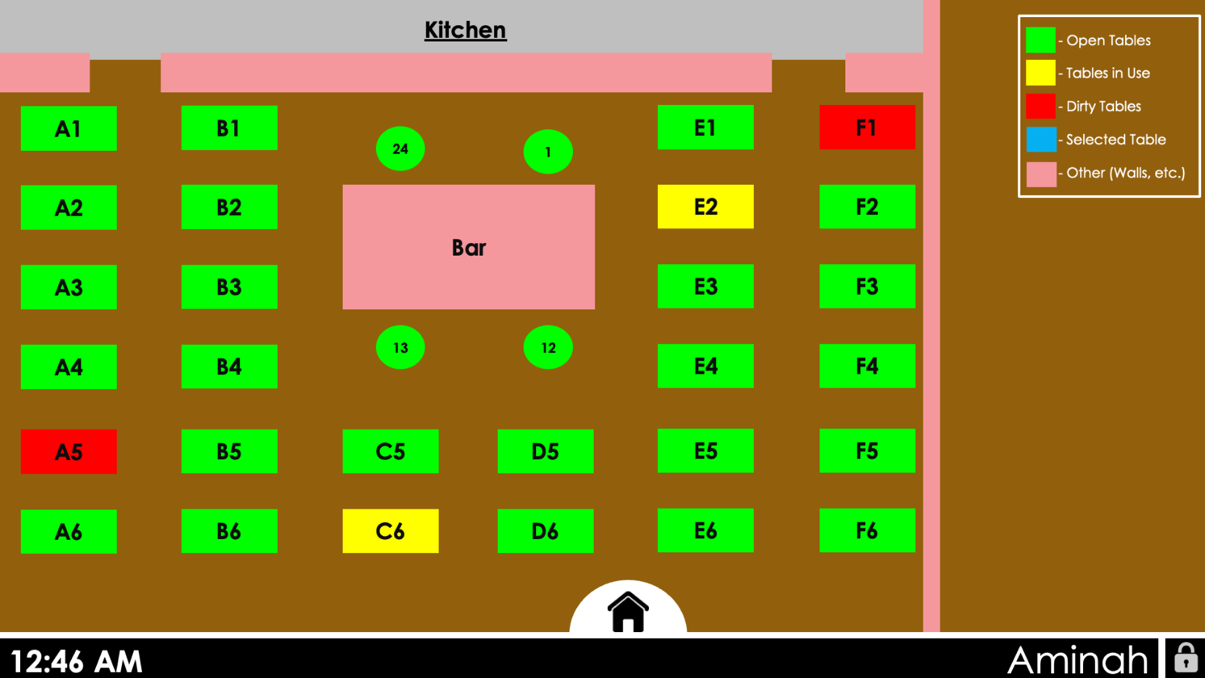
Description automatically generated*Graphical user interface

Description automatically generated with low confidence*Graphical user interface, text, application, chat or text message

Description automatically generated *Figure e: Empty Cart Error Figure f: Drink Menu*

*Figure g: Menu Item Options Figure h: Order Summary*

## The Table View Screen



*Graphical user interface, application

Description automatically generatedGraphical user interface, application

Description automatically generated*Accessible to the Manager, Host/Hostess, Waiter/Waitress, and Busboy. This screen will allow for the user to select a table in which they would like to service. The **Host/Hostess** is exclusively able to select a clean table for customers, which they will then escort those customers to said table. The **Waiter/Waitress** is exclusively able to select a table from which they would like to take orders from. This will prompt them to the Menu Screen, from which they will be able to complete said task. **Waiters** are also able to make tables as dirty, in which doing so will alert the Busboy to complete their task. **Busboys** are exclusively able to mark formerly dirty tables as clean. The Manager can perform all tasks. Below, you will see a few screens which a response to actions taken by the user in will create a prompt within the screen.

*Figure i: Order Submitted Screen Figure j: Clean Table Redundancy Error*

## The Orders Screen

Graphical user interface, text, application, Teams

Description automatically generated

Graphical user interface, text, application, chat or text message

Description automatically generated*Graphical user interface, text, application

Description automatically generated*Accessible to the Cook. This screen will allow for the user to select an order and mark it as complete. This will alert the Waiter assigned to the order, prompting them to deliver the order to the customer. This screen will also alert the user if there is a more recent order present. Below, you will see various possible screens from the Orders Screen.

*Figure k: Order Recency alert Figure l: Order Complete Screen*

## The Employees Screen

Graphical user interface, application, Teams

Description automatically generated

Graphical user interface, application

Description automatically generatedGraphical user interface, text, application, Teams

Description automatically generated Accessible to the Manager. This screen will allow for the user to create and edit employees that work for a particular store. The user will be prompted to enter a Name, Employee Type, and Login Password. If the password must be deemed ‘too simple’ by the program, it will not allow you to create the employee. Below, you will see various possible screens from the Employee Screen

*Figure m: Employee Creation Screen Figure n: Password Simplicity Error*

Diagram

Description automatically generatedFull UML Diagram

A picture containing diagram

Description automatically generatedText

Description automatically generated with low confidenceCloser look at the Diagram:

## Supporting Text Specification

Employee Class

public class Employee

|  |  |  |
| --- | --- | --- |
| **Constructor** | **Description** | |
| Employee() | Default constructor that initializes newly created employee object but doesn’t give any values. | |
| Employee(string ID, string name, char mid, string surname, int gen, int type) | Overloaded constructor that replaces the EmployeeID, FirstName, MiddleInitial, LastName, and type variables with parameters given. Gender will be set to true or false based of the number provided (0 – Male, 1 – Female). Restricted will automatically be set to false. | |
| **Field summary** | | |
| **Modifier and Type** | **Field** | **Description** |
| private string | EmployeeID | A string that is unique to employees and that allows them to login. |
| FirstName | A string indicating the user’s first name. |
| LastName | A string indicating the user’s last name. |
| private char | MiddleIntital | A character indicating the middle initial of the user. |
| private bool | Gender | A Boolean value used to determine Gender (true for male and false for female). |
| Restricted | A Boolean value used to restrict accounts (this will usually be false. |
| Clocked | A Boolean value used to check if employees are clocked in or not. |
| private int | EmployeeType | A int used to determine the type of the employee (and their level of accessibility) (1- Host Staff | 2 - Wait Staff | 3 - Cooks | 4 - Bartenders | 5 - Bussers | 6 – Managers). |
| private double | Hours | Used to track how many hours the employee has worked. |
| private DateTime | clockInTime | Establishes what time the user clocked in. |
| **Method summary** | | |
| **Modifier and Type** | **Method** | **Description** |
| string | getEmployeeID() | Returns the Employees ID number (which also functions as their login info). |
| getFirstName() | Returns the first name of the employee. |
| getLastName() | Returns the surname of the employee. |
| RestrictedAccount() | Returns text saying “restricted” or “unrestricted”. |
| getRole() | Returns text of employee’s job based on employee type and gender. |
| char | getMiddleInitial() | Returns the first letter of the employee’s middle name. |
| bool | getGender() | Checks gender identification. Returns true for male and false for female. |
| isRestricted() | Checks to see if employee’s account is restricted. Returns true for restricted and false for unrestricted. |
| isClocked() | Returns true or false depending on if the user is clocked in or not. |
| void | setEmployeeID(string ID) | Changes EmployeeID to given parameter. |
| setFirstName(string name) | Changes FirstName to string provided in the parameter. |
| setMiddleName(char mid) | Changes MiddleInitial to char provided in the parameter. |
| setLastName(string surname) | Changes LastName to string provided in the parameter. |
| setGender(string MF) | Uses int provided in the parameter to determine Gender (M – Male, F – Female) and assigns it to the according variable. |
| Restrict() | Changes employee’s restricted variable to true, restricting their account. |
| Unrestrict() | Changes employee’s restricted value to false, unrestricting their account if it wasn’t already |
| setEmployeeType(int type) | Changes employee’s type based on the provided parameter. |
| addHours(double time) | Adds a given amount of time to current hours worked. |
| clockIn() | Sets employee to clocked in and sets clocked in time to current time. Then displays a welcome message. |
| clockOut() | Sets employee to clocked out, uses current time, and clocked in time to get number of hours worked for the day and adds it to total amount of hours worked. Then displays a goodbye message. |
| getSummary() | Presents a summary of the employee including the full name, gender, role, total hours worked, and restricted status. This method functions similarly to a ToString method. |

Staff Class

public class Staff

|  |  |  |
| --- | --- | --- |
| **Constructor** | **Description** | |
| Staff() | Initializes a newly created staff object. | |
| **Field summary** | | |
| **Modifier and Type** | **Field** | **Description** |
| public string | Password | String used to create password for employees which is set back to null after employee has been added. |
| Name | String used to create first name for employees which is set back to null after employee has been added. |
| Surname | String used to create last name for employees which is set back to null after employee has been added. |
| public char | Mid | Char used to create middle initial for employees which is set back to null after employee has been added. |
| public int | Gen | Int used to indicate gender (0 – male, 1 – female) which is set back to null after employee is added. |
| count | Used to determine if password is valid or not. If an invalid section is detected the count increases. Too many increases to this value makes the password invalid. |
| type | Int used to indicate employee’s role which is set back to null after employee is added. |
| public char [] | decomposedPassword | Char array to make sure passwords meet all criteria. This is set back to null after employee is created |
| List<Employee> | Roster | A list/arraylist used to keep track of employees. |
| **Method summary** | | |
| **Modifier and Type** | **Method** | **Description** |
| bool | validPassword(string pass) | Inputs a password and checks all criteria to make sure it is acceptable. Then returns true (if password is acceptable) or false (if criteria hasn’t been met). Criteria includes proper length, not too many incremental values, not too many repeating numbers, password is unique. |
| void | addEmployee() | Adds employee to Roster list. This is a function that only a manager can access. |
| removeEmployee() | Removes employee from Roster list. This is a function that only a manager can access. |
| restrictEmployee() | Restricts employees from accessing other functions. This is a function that only a manager can access. |
| unrestrictEmployee() | Unrestricts employees from accessing other functions. This is a function that only a manager can access. |
| Login() | Allows employees to login and perform tasks. |
| ClockInOut() | Allows employees to clock in and tracks time until employee clocks out. Or clocks employee out. |

**Item Class**

public class Item

|  |  |  |
| --- | --- | --- |
| **Constructor** | **Description** | |
| Item(string Name, float Cost) | Creates an item that sets name and price. | |
| Item(string Name, float Cost, Item Additional Item) | Creates an item and adds additional item to go along with it. | |
| **Field summary** | | |
| **Modifier and Type** | **Field** | **Description** |
| private string | Name | String that names the item. |
| private float | Cost | Float that represents the price of the item. |
| private Item | AdditionalItem | Another item that goes along with the base item (ex. Sides, drinks, combos, etc.) |
| **Method summary** | | |
| **Modifier and Type** | **Method** | **Description** |
| float | getCost() | Returns the float value of cost. |
| string | getName() | Returns the name of the item. |
| ToString() | Returns a description of the item by overriding the existing ToString method. |
| void | setAdditionalItem(Item item) | Sets another item to an existing item. |

**Table Class**

public class Table

|  |  |  |
| --- | --- | --- |
| **Constructor** | **Description** | |
| Table(string TableID) | Creates a table class using the Table ID variable. | |
| **Field summary** | | |
| **Modifier and Type** | **Field** | **Description** |
| private string | TableID | Establishes a specific identification for each table. |
| Server | Designates a server employee for each table. |
| private int | TotalPatronSeated | Indicates how many persons were seated at a particular table. |
| private byte | State | Indicates whether a table is open (state 1)(green), unoccupied but dirty (state 2)(red), occupied (state 3)(yellow), or unavailable (state 4)(grey). |
| private DateTime | SeatedTime | Creates a seating time for table. |
| **Method summary** | | |
| **Modifier and Type** | **Method** | **Description** |
| string | getTableID() | Returns the string value which is the table number. |
| PatronsLeave() | Changes state of the table to dirty and returns the turnaround time. |
| getServer() | Returns server name. |
| int | getState() | Returns an int that tells user what state the table is in. This is mainly used to alter the table color. |
| void | SeatCustomer() | Sets table to occupied state and changes SeatedTime to the current time. |
| setServer(string Server) | Changes the server to the one in the parameter. |
| TableCleaned() | Resets table state to clean. |

**Menu Class**

public static class Menu

|  |  |  |
| --- | --- | --- |
| **Constructor** | **Description** | |
| N/A | | |
| **Field summary** | | |
| **Modifier and Type** | **Field** | **Description** |
| private static List<Appetizer> | AppetizerList | Lists/Arraylists containing various items of their respective category. |
| private static List<Salad> | SaladList |
| private static List<Entree> | EntreeList |
| private static List<Sandwiche> | SandwicheList |
| WrapList |
| BurgerList |
| private static List<Drink> | DrinkList |
| private static List<Side> | SideList |
| string | line | String used to locate where StreamReader is in text file. |
| int | counter | Value to indicate what number line StreamReader is on. |
| **Method summary** | | |
| **Modifier and Type** | **Method** | **Description** |
| static void | SetUpList(string FilePath) | Takes in a text file and sets up list with given file. |
| PrintList<T>(List<T> List) where T: Item | Takes in a List of any variable as long as said variable is child of item class (Appetizer, Salad, Entrée, Sandwich, Drink, Side) and prints the respective item.ToString method. |
| PrintMenu() | Prints every List in the menu class. |
| static Item | FindItem <T> (List<T> List, string x) where T : Item | Searches a specific menu for an item and returns it if it’s found. |
| GetItem(string Name) | Uses the FindItem method to search entire menu for an item by name and returns it if it’s found. |

**Order Class**

public class Order

|  |  |  |
| --- | --- | --- |
| **Constructor** | **Description** | |
| Order() | Creates a placed time and calculates total cost. | |
| Order(string TableID) | This constructor creates and order similar to the above method but assigns it to a table. | |
| Order(string TableID, ArrayList list) | Overloaded Constructor that assigns the order to a table and allows them to order multiple items. | |
| **Field summary** | | |
| **Modifier and Type** | **Field** | **Description** |
| private ArrayList | ItemList | Creates an ArrayList to store the various items for the order. |
| private DateTime | PlacedTime | Designates the time that the order was placed. |
| private float | TotalCost | Designates the total cost of the order. |
| private string | TableID | Assigns order to a table. |
| private static int | Counter | Designates a unique value and gives it to the order. |
| private int | OrderNumber | Creates a value and gives it to the order. |
| **Method summary** | | |
| **Modifier and Type** | **Method** | **Description** |
| float | CalculateTotalCost() | Returns a float of the total value of all items. |
| getTotalCost() | Returns the total cost. |
| void | AddItem(Item AddedItem) | Adds a new item into the ItemList and adds it to the total. |
| RemoveItem(int index) | Removes an item from the list and subtracts it from the total cost. |
| int | getOrderNumber() | Returns the order number. |
| DateTime | GetPlacedTime() | Returns the time that the order was placed. |
| bool | isEmpty() | Returns true if ItemList is empty and false otherwise. |

Database table descriptions

|  |  |  |  |
| --- | --- | --- | --- |
| **TableName** | **PrimaryKey** | **ForeignKey** | **Attributes** |
| Employee | EmployeeNumber |  | Gender |
| EmployeeList | EmployeeNumber |  | FirstName, MiddleInitial, LastName |
| Table | TableID | OrderNumber | TotalPatronSeated, State |
| Menu | ItemName |  | AppetizerList, SaladList, EntreeList, SandwichList, WrapsList, BurgerList, DrinksList, SideList |
| Item | ItemName |  | Cost |
| Order | OrderNumber | TableID | TotalCost, PlacedTime |

## Technical Support Specification

All Managers will have a basic training which will cover exactly what permissions each employee type will have, all screens and frequently occurring errors in each screen, program installation on all company-owned devices, system troubleshooting tips, and more. This training will be accompanied by a comprehensive user manual with detailed information about

All employees must complete a two-hour training course which will verify that they have a fundamental understanding of all positions in the restaurant. Employees will also be provided with documentation detailing all information regarding their position in relation to the application.

# Appendix

Hours Spent Outside of class:

2 hours per week

Total: 30 hours