We are very pleased to announce our “VecPOS” system to serve the purpose of an integrated Point of Sale System ready to be used in your restaurants. (Ignore my attempt to advertise so monotonically :D) I would like to refer to the system built in this document as VecPOS.

Software Design for “VecPOS” System

The software suite for the VecPos system would be pre-deployed to the POS hardware used in restaurant typically computer(s), tablets etc. Our direct customers would be Restaurant manager(s), servers. The POS systems could be given to each server apart from manager and it is thus required for the devices to communicate with a shared server (master-slave architecture) and database.The data communicated must be consistent, synchronized and fault tolerant.

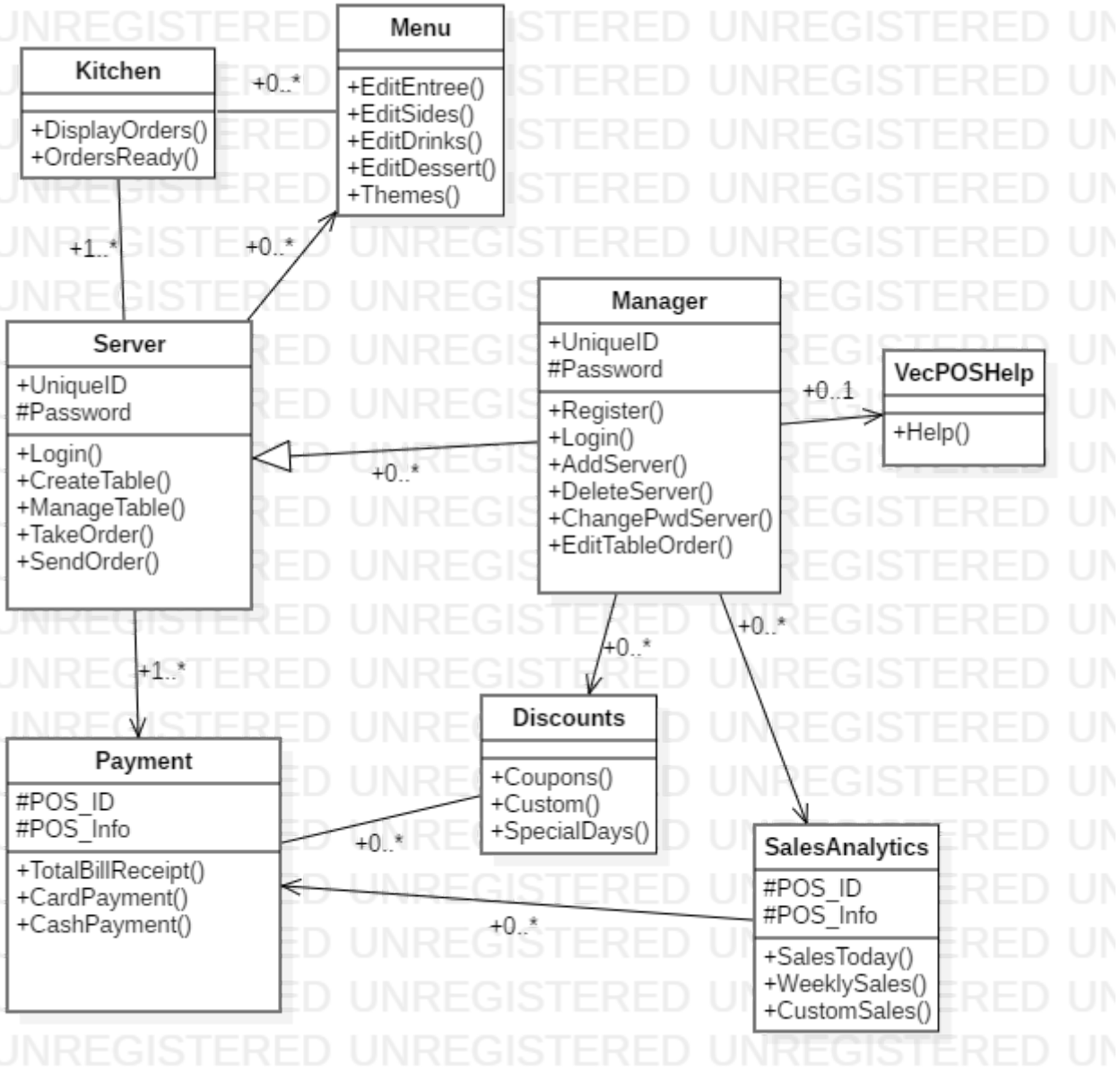
Let us provide a domain name or IP Address for our users, the restaurant management. The welcome page must consist of a link to the restaurant managers’ registration page, FAQ/Help/Support links to contact Vecstone, login, password fields. The login and password fields are common to managers and servers. In our backend, we assign specific functionalities only to the managers by maintaining their user IDs separately and validation is done as soon as the login button is clicked. This privilege access to managers allows them to add or delete servers, change servers passwords, apply discounts to tables, edit orders of tables etc. provided on their page after successful login.

The servers must find the design easy and quick to use to take orders and process them quickly. It would be convenient if the server could login only once (maintain session) after he/she starts their job and until they take a break or leave for the day. Alternative secure mechanisms like pin login or fingerprint could be provided in addition and for now I am not considering those easy-authentication features but could be extended easily to the auth database.

The challenge mentions a maximum of 4 customers per table, but in my opinion I think it would be extremely beneficial to have a more seating capacity for each table. This must be a direct requirement from the client, but we could see if the tables and seating areas are modular. We could just combine multiple tables which are required to cater the needs of special venues say a wedding dinner rather than not making customers welcoming when they come in group of 5 or more. If our client accepts this proposal, we can make our billing mechanism accordingly with changing the regular layout of tables itself on the software or physical layout fed to system. We just combine bills from multiple tables and provide just one if required by appending all the table data combined for the managers to understand even at a later point of time while revisiting archived bills.

Throughout a day, each table would be used by many customer groups visiting the restaurant and a mechanism to distinguish would be needed. Say table “10/1” is used to indicate the first group of customer who are seated at table number 10. Similarly, “10/2” on the receipt would indicate 2nd group of customers on the same table 10. Timestamp could also be recorded while server assigns a table. Also, if reservations are booked in advance, we can block the respective tables for a certain period of time and can be made automatically available to the empty tables pool, incase if the customers fail to show up.

Below is the initial draft of the Class Diagram, I designed for the proposed POS system “VecPOS”



The class “Menu” is accessed by Server, Manager, Kitchen and is only modifiable by the Kitchen. This makes the Head Chef change items in Menu dynamically. Also there could be themed menus like Sunday Buffet, Easter Theme Fixed Buffet etc, which the Chef designs and publishes. We could thus have some specific functionalities designed for chefs and cooks. Else in the alpha version, the chef could communicate with the manager and he/she could change Menu by providing features.

The “Kitchen” class could display all the items that could be ordered. Once items for a table are ready, the OrdersReady() function could be called by Chef which would notify the server of a respective table the food is ready for a particular table to be served directly to the server’s device if given one.

The “Tables” class could have name of the restaurant as a field (incase our client opens a new chain of restaurant and wants to expand our VecPOS as well) and DisplayLayout() could show the simple graphical architecture of tables and labeled table numbers. The tables could have a label on them or a color representation to distinguish between Unoccupied, reserved and seated customers.

This would also imply the actions that a server can take by clicking a particular table and performing actions like allocating a table for the customers (AssignTable()), cleared table available back to pool among our Tables resources (ClearTable()) and blocking a table for a certain period if there is a reservation.

The “Manager” as indicated before could login to the portal with their unique userID and encrypted password. The Manager has functions to Add or Delete servers, edit orders of a table already sent to kitchen and change passwords of servers whenever required. For this, we first allow the manager ID’s to be registered in our database and distinguish them by providing special roles. The Manager(s) may first register using a link we provide either on the login page or via a link to manager approved personal email to perform setup/registration. We must provide support to atleast manager(s) in our first version of software. They could have a portal to raise tickets/complaints for any issue they might face.

The “Server” class comprises of userID and Password as fields. We could provide easy login via PIN option and could even maintain session without the need to login multiple times a day/shift. The server could take orders by choosing from the menu (1 entree and 2 sides) by TakeOrder() and send it to kitchen via SendOrder().

The “Payment” class could be called by any of the object of “Server” or “Manager”. Once chosen for a single customer or multiple customers of a table, it fetched the prices of items ordered from menu and sums them up. The percentage tax and a default gratitude (20% given) or custom gratitude could be added up to make the total amount and itemized receipt could be printed. This receipt could contain several details as per the requirements of management and how they wish to process the data. When cash or card is provided to server, the server could take it up to manager to access the card debit machine or cash drawer.

The Manager could also provide discounts to the customers or table orders. The “Discounts” class is accessed by manager(s) to provide custom discount or several other type of redemptions, promotions while bill payment.

In view of the business models, tracking sales, increasing revenues, understanding customer behaviour and several other analytics we could provide a “SalesAnalytics” class. Models could be generated which get more and more richer with more data as time progresses. Beginning from our initial version, we can start collecting data of receipts, use OCR technologies and text analysis.

We could also design our system, where manager also has all the roles of a server (Manager class inherits Server class). This could allow manager to run restaurant independently.
The ID, Password, Menu details could be stored in database server. However, if the setup is confined to a just within the restaurant, we can store them on local database and provide DB authentication mechanisms. The passwords will be encrypted and Card data if stored for few minutes to deduct tip/gratuity amount should be encrypted as well. The food selection menu must contain images of dishes grouped category wise. The items must be accompanied by short description and ingredients in it. Allergy indications must be displayed upon customers request or atleast made known to them.

Author: Murali Kammili
mkammili@iu.edu