Restaurant Automation System
Group Members:
Ian Ramazan, George Dzagali, James Schaller

Project Description:

The restaurant world is a competitive one, and we feel that our best chance to get ahead is to fully automate our restaurant operations. This would allow for customers to have smoother transitions from the time they walk through the door to when they pay the bill.

We would like to be able to cut down on the time it takes for a customer to place an order, and the chef to receive it. In order to do this, we would like our servers to be able to directly send orders to our chefs, which would help to get a customer's order out as quick as possible. On top of this, we require a system to prioritize orders based on the time that they were placed, and a system that would allow our chefs to signal the status of an order as being "stopped", "in progress", or "completed", and would show the time since the dish started being prepared, so that the server can have a rough estimate on how long it will take to complete. This would allow for our servers to not have to constantly check in with the kitchen to check the status of their orders.

Everyone who has been to a restaurant knows that paying the bill can be tedious and downright annoying. To solve this, we would like our customers to be able to pay the bill directly from the table after they are finished eating. They would also be able to split the bill between the number of people at the table. This allows for a smooth transition from the end of their meal, to paying the bill, to leaving as happy and satisfied customers.

Physical menus are an archaic practice, and we wish to move away from it. In order to do this we would like an app that our customers can use to place orders that are directly sent to the kitchen, so that there is no wait from the time they are seated.

Aside from customer interactions and order placement, we would also like the app to be able to help our managers with their duties. We would like our managers to be able to view different profit margins, store expenses, and payrolls. This would help lighten the load on our managers, so that they can be more active with the restaurant itself. Our managers should also be able to add or remove items from the menu based on availability.

In addition, we want to be able to have our employees have immediate access to the resources they require. This means that we would like the app to include an employee portal, that would be able to show work shifts, and to include a scheduling manager.

System Requirement:

Identifier	**Priority**	**Requirements**
REQ1	5	● The system shall keep itself passcode locked at all times, unless instructed otherwise by an authorized user.
REQ2	2	● The system shall then return it's locked state by pressing an exclusive button.
REQ3	3	● The system should allow users to add, manage and remove customers from a table
REQ4	4	● The system should maintain all categories of available items that are necessary for everyday use
REQ5	4	● The system should generate a checkout functionality for specified tables to create a bill.
REQ6	4	● The system should allow for making and monitoring orders.

User Stories:

Identifier	User Stories	Size
ST-1	As a restaurant owner, I can monitor the supply of ingredients in my restaurant.	4 pts
ST-2	As a chef, I can more easily assess when I should be preparing a certain dish.	3 pts
ST-3	As a server, I can more efficiently manage and view information on the tables I am responsible for.	2 pts
ST-4	As a restaurant owner, I can view financial data.	6 pts
ST-5	As a chef, I can see the popularity of various dishes.	3 pts
ST-6	As a customer, I can immediately place my order when I am ready to do so.	2 pts
ST-7	As a restaurant owner, I can view employee data.	2 pts
ST-8	As a server, I have access to information on the status of the dishes my tables have ordered.	3 pts
ST-9	As a customer, I can pay my bill when I am ready without intervention from the restaurant staff.	3 pts
ST-10	As a server, I can manage my shifts and view payment	2 pts

	information.	
ST-11	As a customer, I can make reservations.	2 pts

Use Cases:

Actor	Actor’s Goal	Use Case Name
Customer	Pay a bill.	PayBill(UC-1)
Customer	Place orders and make replacements if wanted.	PlaceOrder(UC-2)
Server	Manage and view information on tables.	Table(UC-3)
Owner	View and edit the stock of ingredients	Stock(UC-4)
Owner	View and edit menu items	Menu(UC-5)
Owner, Server	Access the system with a employee-specific pin.	Authenticate(UC-6)

Traceability Matrix:

Req’ts	UC1	UC2	UC3	UC4	UC5	UC5
REQ1						X
REQ2						X
REQ3			X			
REQ4				X	X	
REQ5	X					
REQ6		X				

Use Case UC-1: PayBill

Related Requirements:	REQ5
Initiating Actor:	Any of: Customer
Actor's Goal:	Pay a Bill
Participating Actors:	Bill, Table, Order
Preconditions:	Seated table has no outstanding orders.
Postconditions:	The table is now open and dirty.

Flow of Events for Main Success Scenario:

-> 1. Customer requests a Bill

<- 2. System looks at orders for the table and displays a bill for those orders

Use Case UC-2: PlaceOrder

Related Requirements:	REQ6
Initiating Actor:	Any of: Customer
Actor's Goal:	Place an order assigned to a table.
Participating Actors:	Table, Order
Preconditions:	A table is seated.
Postconditions:	The table has an additional order assigned to it.

Flow of Events for Main Success Scenario:

-> 1. Customer makes a request for an order

<- 2. System places that order in the list of orders for the table and the restaurant

Use Case UC-3: Table

Related Requirements:	REQ3
Initiating Actor:	Any of: Server
Actor's Goal:	View information on tables in the Restaurant.
Participating Actors:	Server, RestaurantData, Table
Preconditions:	There are tables in the restaurant
Postconditions:	N/A

Flow of Events for Main Success Scenario:

-> 1. Server chooses a table to view

<- 2. System displays information on the chosen table

Use Case UC-4: Stock

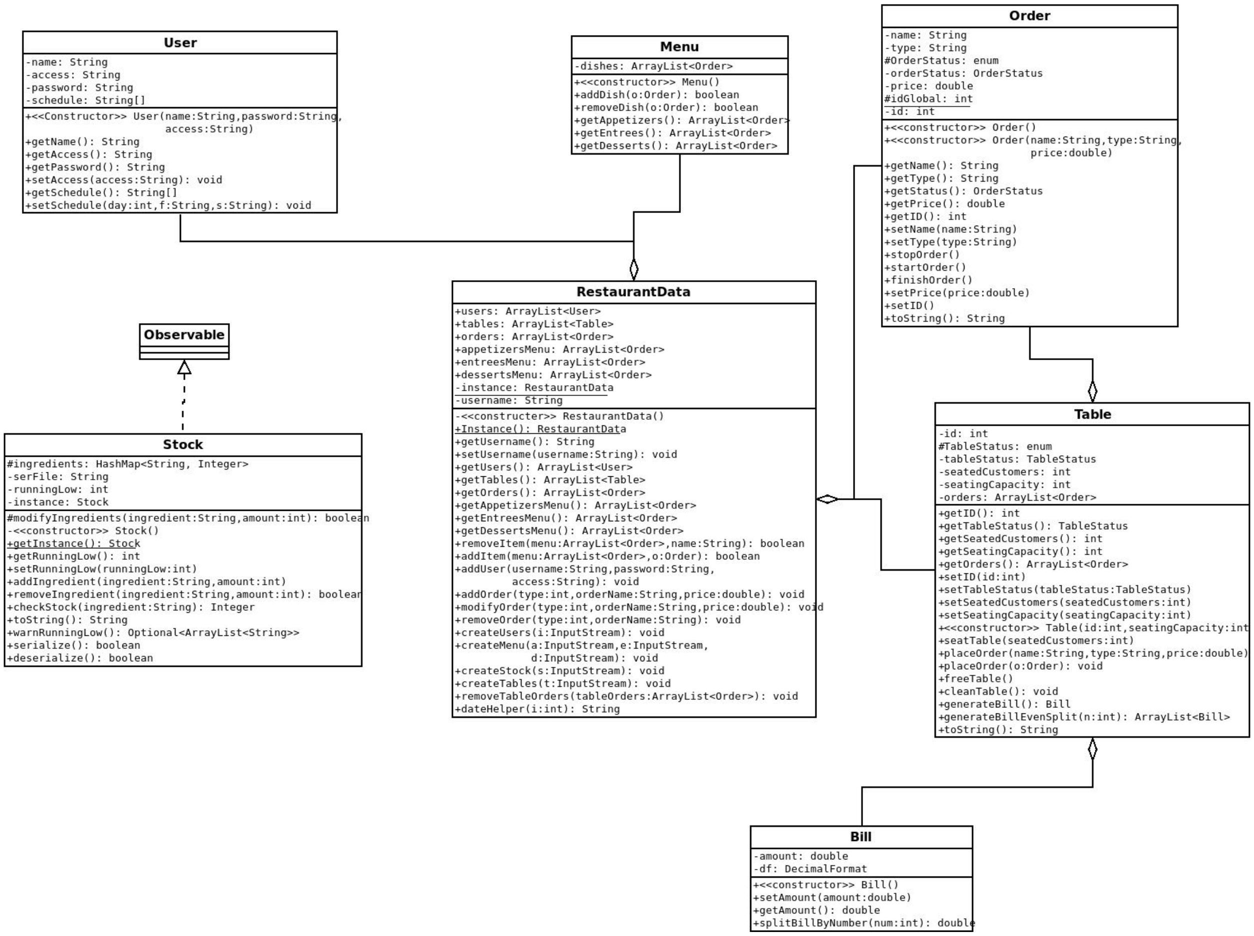
Related Requirements: REQ4
Initiating Actor: Any of: Owner
Actor's Goal: View stock of ingredients in the restaurant
Participating Actors: User, Stock
Preconditions: Owner is logged in
Postconditions: Stock of ingredients has been changed
Flow of Events for Main Success Scenario:
-> 1. Owner requests list of ingredients
<- 2. System displays ingredients list
-> 3. Owner requests an edit of the ingredients stock
<- 4. System edits the restaurant stock instance

Use Case UC-5: Menu

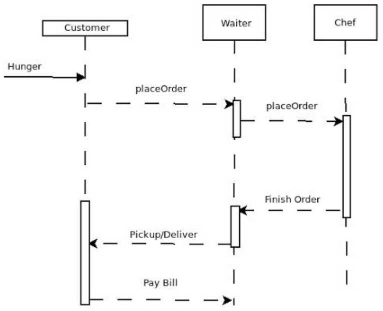
Related Requirements: REQ4
Initiating Actor: Any of: Owner
Actor's Goal: Access and make edits to the restaurant's menu
Participating Actors: Menu, RestaurantData, User
Preconditions: Owner is logged in
Postconditions: Menu has been edited
Flow of Events for Main Success Scenario:
-> 1. Owner makes a request to view the menu
<- 2. System fetches and displays the menu
-> 3. Owner requests changes to the menu
<- 4. System registers changes made to the menu in RestaurantData

Use Case UC-6: Authenticate

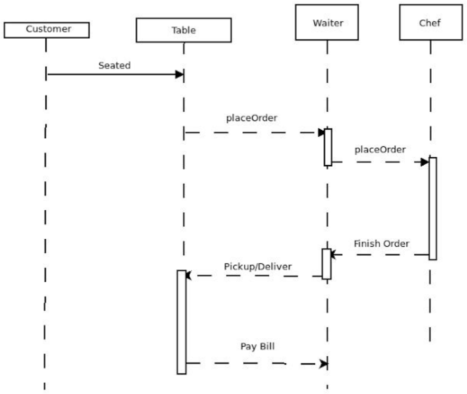
Related Requirements: REQ1 and REQ2
Initiating Actor: Any of: Owner, Server
Actor's Goal: Access employee-specific funcionality using a username and Password
Participating Actors: User
Preconditions: A user is not logged in
Postconditions: A user is logged in
Flow of Events for Main Success Scenario:
-> 1. User makes a login request
<- 2. System checks the user login request for validity
3. User is logged in
Flow of Events for Extensions (Alternate Scenario):
<- 2a. System find the login request is invalid
3a. User login is rejected


User
-name: String
-access: String
-password: String
-schedule: String[]
+<<Constructor>> User(name:String,password:String, access:String)
+getName(): String
+getAccess(): String
+getPassword(): String
+setAccess(access:String): void
+getSchedule(): String[]
+setSchedule(day:int,f:String,s:String): void
Menu
-dishes: ArrayList<Order>
+<<constructor>> Menu()
+addDish(o:Order): boolean
+removeDish(o:Order): boolean
+getAppetizers(): ArrayList<Order>
+getEntrees(): ArrayList<Order>
+getDesserts(): ArrayList<Order>
Order
-name: String
-type: String
#OrderStatus: enum
-orderStatus: OrderStatus
-price: double
#idGlobal: int
-id: int
+<<constructor>> Order()
+<<constructor>> Order(name:String,type:String, price:double)
+getName(): String
+getType(): String
+getStatus(): OrderStatus
+getPrice(): double
+getID(): int
+setName(name:String)
+setType(type:String)
+stopOrder()
+startOrder()
+finishOrder()
+setPrice(price:double)
+setID()
+toString(): String
RestaurantData
+users: ArrayList<User>
+tables: ArrayList<Table>
+orders: ArrayList<Order>
+appetizersMenu: ArrayList<Order>
+entreesMenu: ArrayList<Order>
+dessertsMenu: ArrayList<Order>
-instance: RestaurantData
-username: String
-<<constructer>> RestaurantData()
+Instance(): RestaurantData
+getUsername(): String
+setUsername(username:String): void
+getUsers(): ArrayList<User>
+getTables(): ArrayList<Table>
+getOrders(): ArrayList<Order>
+getAppetizersMenu(): ArrayList<Order>
+getEntreesMenu(): ArrayList<Order>
+getDessertsMenu(): ArrayList<Order>
+removeItem(menu:ArrayList<Order>,name:String): boolean
+addItem(menu:ArrayList<Order>,o:Order): boolean
+addUser(username:String,password:String, access:String): void
+addOrder(type:int,orderName:String,price:double): void
+modifyOrder(type:int,orderName:String,price:double): void
+removeOrder(type:int,orderName:String): void
+createUsers(i:InputStream): void
+createMenu(a:InputStream,e:InputStream, d:InputStream): void
+createStock(s:InputStream): void
+createTables(t:InputStream): void
+removeTableOrders(tableOrders:ArrayList<Order>): void
+dateHelper(i:int): String
Observable
Stock
#ingredients: HashMap<String, Integer>
-serFile: String
-runningLow: int
-instance: Stock
#modifyIngredients(ingredient:String,amount:int): boolean
-<<constructor>> Stock()
+getInstance(): Stock
+getRunningLow(): int
+setRunningLow(runningLow:int)
+addIngredient(ingredient:String,amount:int)
+removeIngredient(ingredient:String,amount:int): boolean
+checkStock(ingredient:String): Integer
+toString(): String
+warnRunningLow(): Optional<ArrayList<String>>
+serialize(): boolean
+deserialize(): boolean
Table
-id: int
#TableStatus: enum
-tableStatus: TableStatus
-seatedCustomers: int
-seatingCapacity: int
-orders: ArrayList<Order>
+getID(): int
+getTableStatus(): TableStatus
+getSeatedCustomers(): int
+getSeatingCapacity(): int
+getOrders(): ArrayList<Order>
+setID(id:int)
+setTableStatus(tableStatus:TableStatus)
+setSeatedCustomers(seatedCustomers:int)
+setSeatingCapacity(seatingCapacity:int)
+<<constructor>> Table(id:int,seatingCapacity:int
+seatTable(seatedCustomers:int)
+placeOrder(name:String,type:String,price:double)
+placeOrder(o:Order): void
+freeTable()
+cleanTable(): void
+generateBill(): Bill
+generateBillEvenSplit(n:int): ArrayList<Bill>
+toString(): String
Bill
-amount: double
-df: DecimalFormat
+<<constructor>> Bill()
+setAmount(amount:double)
+getAmount(): double
+splitBillByNumber(num:int): double

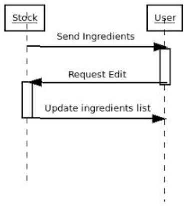
Customer Order



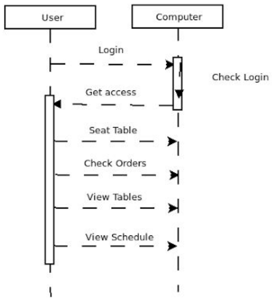
Seated Customer



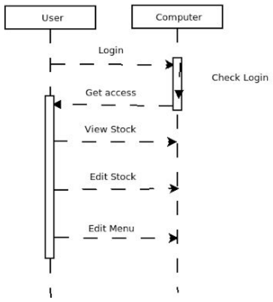
Owner Edit Stock



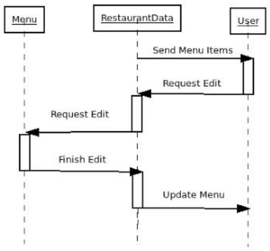
Waiter/Chef Login



Manager/Owner Login



Owner Edit Menu


Menu
RestaurantData
User
Send Menu Items
Request Edit
Request Edit
Finish Edit
Update Menu