GravyXpress:

A Restaurant Management Software

Group Number: #4

Team Members

Name	Email
Yehuda Cohen	yehuda.cohen@rutgers.edu
Shivani Sethi	shivani.sethi@rutgers.edu
Abdul Rattu	r.abdulsami@gmail.com
Amizan Jaleel	najm555@gmail.com
Nabil Ali	alinabil07@gmail.com
Rohit Lakshmanatirthakatte	rohit.lakshmana@rutgers.edu

Instructor: Prof. Ivan Marsic

Project URL: http://gravyxpress.appspot.com/

Revision History:

Version No.	Date of Revision
1	03/03/2013
2	03/10/2013
3	03/17/2013

Individual Contributions:

	Abdul Rattu	Amizan Jaleel	Nabil Ali	Rohit Lakshmana	Shivani Sethi	Yehuda Cohen
Sec. 1: Interaction Diagrams (30 points)	16.67% (5pts)	16.67% (5pts)	16.67% (5pts)	16.67% (5pts)	16.67% (5pts)	16.67% (5pts)
Sec. 2: Classes + Specs (10 points)	50% (5pts)					50% (5pts)
Sec. 3: Sys Arch and Design (15 points)	13.3% (2pts)	33.33% (5pts)		10% (1.5pts)	38.33% (5.75pts)	5% (.75pts)
Sec. 4: Algorithms and Data Structures (4 points)			37.5% (1.5pts)		62.5% (2.5pts)	
Sec. 5: User Interface (11 points)	18.18% (2pts)	36.36% (4pts)			22.7% (2.5pts)	22.7% (2.5pts)
Sec. 6: Testing Design (12 points)			45.83% (5.5pts)	54.17% (6.5pts)		
Sec. 7: Project Management (18 points)	16.67% (3pts)	16.67 (3pts)	16.67% (3pts)	16.67% (3pts)	16.67% (3pts)	16.67% (3pts)

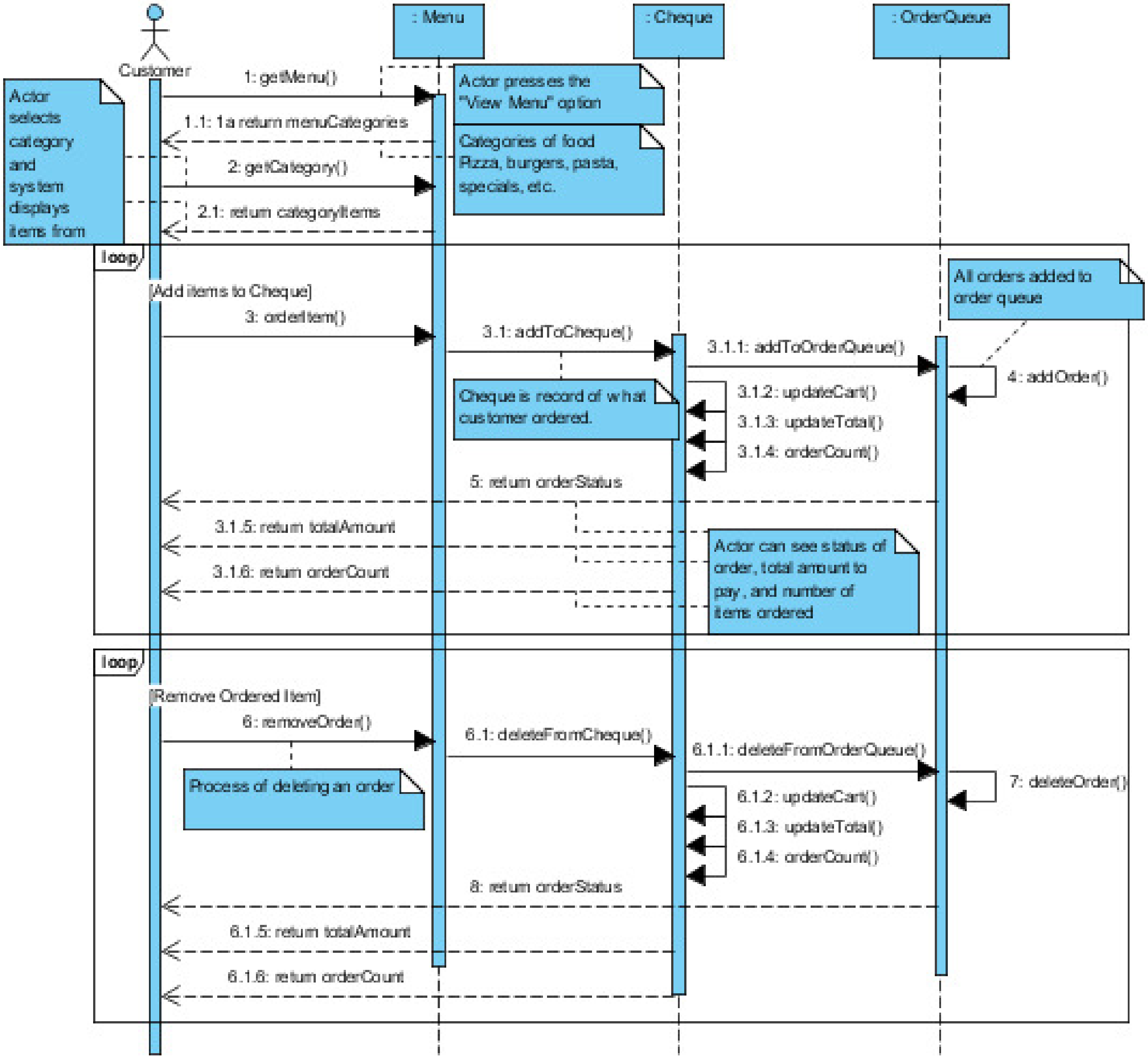
- **Yehuda Cohen**
  - For Part 1, I designed the ServeTable interaction diagram and elaboration. (Also provided a general template for other Interaction Diagrams by completing less detailed versions in Report 1.)
  - For Part 2, I built the class diagram, defining all objects, attributes and methods to make up GravyXpress. In addition, included a very brief description of each class. Furthermore, dealt with the network protocols, and aided Amizan by elaborating some fundamentals of the design of the database.
  -For Part 3, Broke down all responsibilities assigning user stories to all members of the group for development. Tried to ensure the user stories were somewhat related, so members could develop a level of expertese. Additionally, elaborated technical aspects of user interface and user interface implementation.
- **Abdul Rattu**
  - For Part 1, I designed ChangeMenu interaction diagram as well as explanation of functions that have been used within the diagram.
  - For Part 2, I wrote "data types and operational signatures" with detailed descriptions as well as did "mapping subsystems to hardware" part.
  - For Part 3, I have improved the user interface design and contributed in project management.
- **Nabil Ali**
  - For Part 1, I designed ManageOrder interaction diagram and explanation of the diagram.

- For Part 2, I constructed hardware requirements tables and wrote descriptions.
- **Amizan Jaleel**
  - For Part 1, I designed ChangeDrinks interaction diagram as well as explanation using comment bubbles.
  - For Part 2, I did the database design schema as well explanation for it.
  -For Part 3, worked on UI along with Abdul and ocntributed to project management
- **Shivani Sethi**
  - For Part 1, I designed CreateWebpage interaction diagram with explanation. I made significant improvements to the previous diagram in Report 1 and added alternate design ideas.
  - For Part 2, I designed the architectural style of the system and identified and explained all of the subsystems. I drew diagrams to help further explain these sections
  - For Part 3, I did most of the Algorithms and Data Structures Section. I also wrote in half of the UI description.
  -For Project Management, I copied Part 2 work into this final google doc, updated the Table of Contents, setup Google+ hangouts and resolved issues with the distribution of work so that it was more equally distributed
  - I also helped post important updates on the Facebook group page for this Software Group Project
  -I compiled and edited the final version of this report
- **Rohit Lakshmanatirthakatte**
  - For Part 1, I designed OrderFood interaction diagram as well as wrote explanations of some functions using comment bubbles.
  - For Part 2, I made the traceability matrix of all classes using the class diagram and wrote the global control flow for the system architecture and design. For the traceability matrix, I also added and improved upon the domain concepts.
  - For Part 3, I designed all test cases and did 2 unit tests (Cheque and Table).
  - I also wrote half of the Table of Contents, Project Management for Parts 1 and 2 only, and 3/4 of the References section of this report.
  - For each part of the report I opened up a Google Drive document and made announcements about the due date of each deliverable as well as provided links to valuable resources. For parts 1 and 2, I wrote the headings for each section so other team members will know where to write their assigned sections.
  - For part 2, I kept a list of assigned tasks in a notepad and posted it on Google Drive for everyone to see in case they forgot what they were supposed to do. Tasks were assigned by everyone during a Google+ Hangout session.
  - For all parts, I submitted the final pdf document into my Sakai Dropbox.
  - With the idea from Yehuda Cohen, I opened up a Facebook Group for all team members called Software Engineering GravyXpress. The purpose of this group is to allow members to post announcements, decide meeting times, post pictures and program code, and increase communication.
  - I am the only person maintaining and updating the GravyXpress group website http://gravyxpress.appspot.com/ by uploading files, making announcements at each deliverable due date, adding a picture, and adding background color. But the website itself was programmed and created by Yehuda Cohen.

Table of Contents:

1. Interaction Diagrams

OrderFood Use Case:

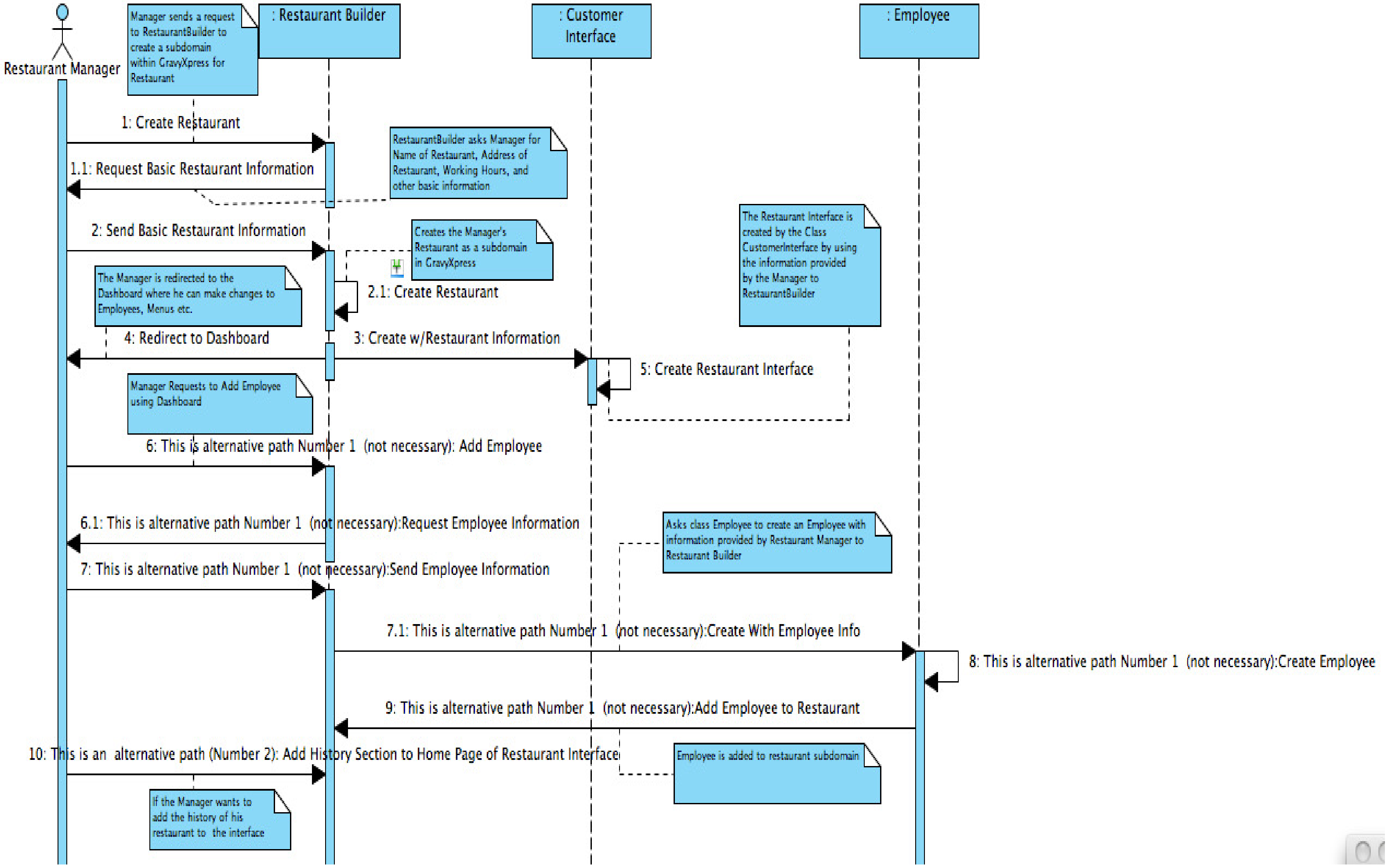

: Menu
: Cheque
: OrderQueue
Customer
Actor selects category and system displays items from
1: getMenu()
Actor presses the "View Menu" option
1.1: 1a return menuCategories
Categories of food Pizza, burgers, pasta, specials, etc.
2: getCategory()
2.1: return categoryItems
loop
[Add items to Cheque]
3: orderItem()
3.1: addToCheque()
3.1.1: addToOrderQueue()
All orders added to order queue
4: addOrder()
Cheque is record of what customer ordered.
3.1.2: updateCart()
3.1.3: updateTotal()
3.1.4: orderCount()
5: return orderStatus
3.1.5: return totalAmount
3.1.6: return orderCount
Actor can see status of order, total amount to pay, and number of items ordered
loop
[Remove Ordered Item]
6: removeOrder()
6.1: deleteFromCheque()
6.1.1: deleteFromOrderQueue()
7: deleteOrder()
Process of deleting an order
6.1.2: updateCart()
6.1.3: updateTotal()
6.1.4: orderCount()
8: return orderStatus
6.1.5: return totalAmount
6.1.6: return orderCount

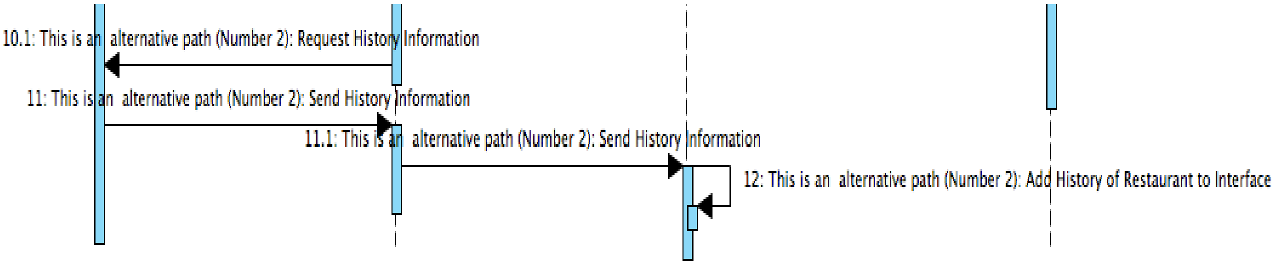
The customer can request the Menu class to see the menu items (organized by categories). It can then send a request to the Menu class to order an item. The Menu class will then send a request to the Cheque class to add the item to the order cheque for the customer. Furthermore, the item will be added to the order queue as the Cheque class will immediately send a request to the OrderQueue class to do so. The Cheque class will send the information about the bill to the customer. To delete an order, the Customer class will send a request to the Menu class which will proceed to tell the Cheque and subsequently the OrderQueue classes to remove the order.

An alternative design that was considered was to have a single Order button at bottom of the list of ordered items. In the interaction diagram above, we have several order buttons, one for each item in the restaurant menu. So, the system automatically sends the order into the order queue. In the alternative design, the user can collect all items he/she wants to order and then send them all simultaneously to the order queue. We chose the automatic submission of orders over the alternative design because the customer might forget to hit the Order button at the end, and then might complain about where his/her orders are even though he/she did not send them to the order queue.

Another alternative design that was considered was having a feature that shows the estimated time of food arrival. However, we realized that this was too difficult to implement, and also there are other factors that can affect the time that the customer gets the food. The ingredients might run out during preparation or there are too few kitchen staff workers or the dish takes a long time to prepare. So, estimated time of food arrival will not be designed and implemented.

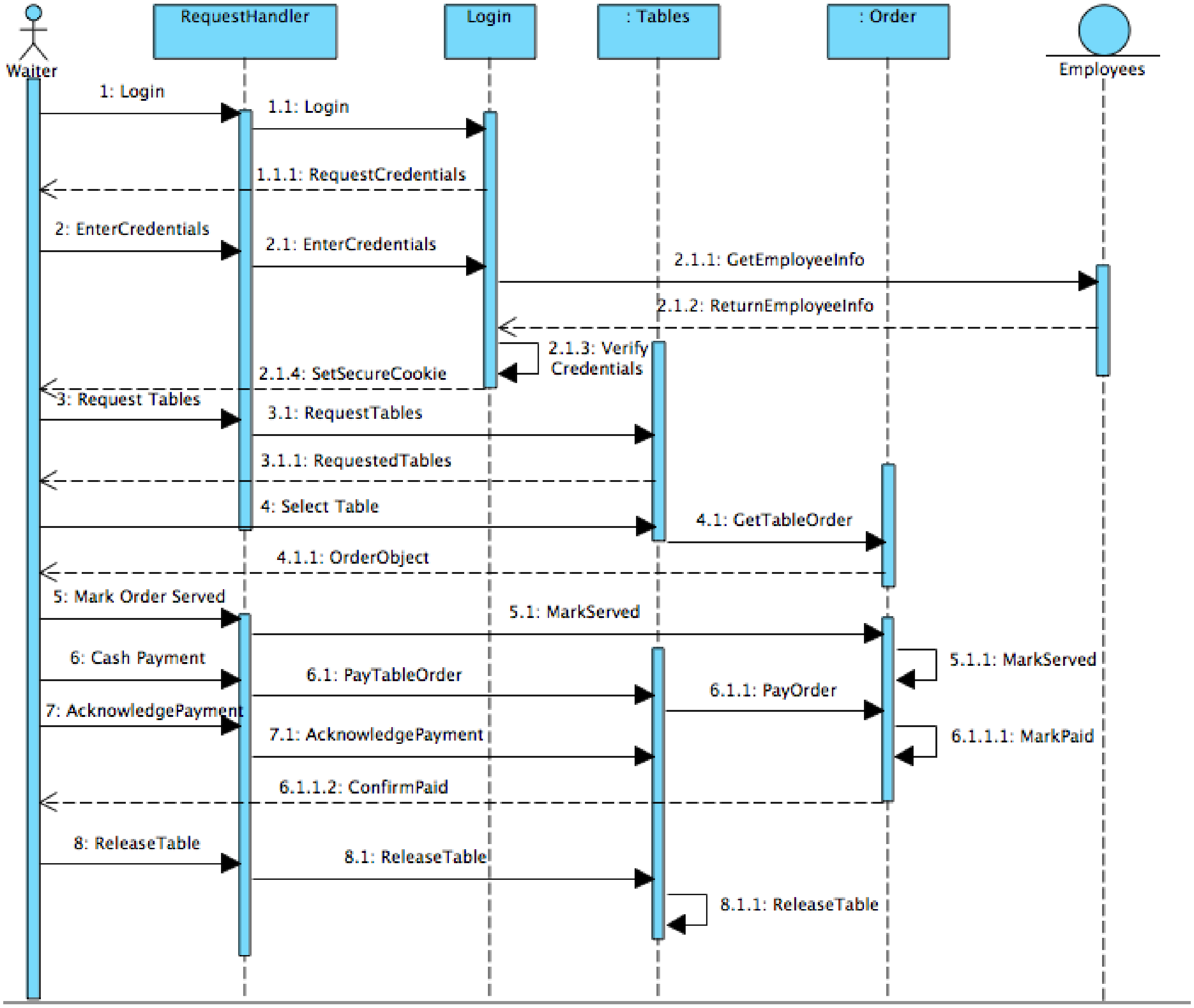
CreateWebpage Use Case:


Restaurant Manager
: Restaurant Builder
: Customer Interface
: Employee
Manager sends a request to RestaurantBuilder to create a subdomain within GravyXpress for Restaurant
1: Create Restaurant
1.1: Request Basic Restaurant Information
RestaurantBuilder asks Manager for Name of Restaurant, Address of Restaurant, Working Hours, and other basic information
2: Send Basic Restaurant Information
Creates the Manager's Restaurant as a subdomain in GravyXpress
The Restaurant Interface is created by the Class CustomerInterface by using the information provided by the Manager to RestaurantBuilder
The Manager is redirected to the Dashboard where he can make changes to Employees, Menus etc.
2.1: Create Restaurant
4: Redirect to Dashboard
3: Create w/Restaurant Information
5: Create Restaurant Interface
Manager Requests to Add Employee using Dashboard
6: This is alternative path Number 1 (not necessary): Add Employee
6.1: This is alternative path Number 1 (not necessary):Request Employee Information
Asks class Employee to create an Employee with information provided by Restaurant Manager to Restaurant Builder
7: This is alternative path Number 1 (not necessary):Send Employee Information
7.1: This is alternative path Number 1 (not necessary):Create With Employee Info
8: This is alternative path Number 1 (not necessary):Create Employee
9: This is alternative path Number 1 (not necessary):Add Employee to Restaurant
10: This is an alternative path (Number 2): Add History Section to Home Page of Restaurant Interface
Employee is added to restaurant subdomain
If the Manager wants to add the history of his restaurant to the interface


10.1: This is an alternative path (Number 2): Request History Information
11: This is an alternative path (Number 2): Send History Information
11.1: This is an alternative path (Number 2): Send History Information
12: This is an alternative path (Number 2): Add History of Restaurant to Interface

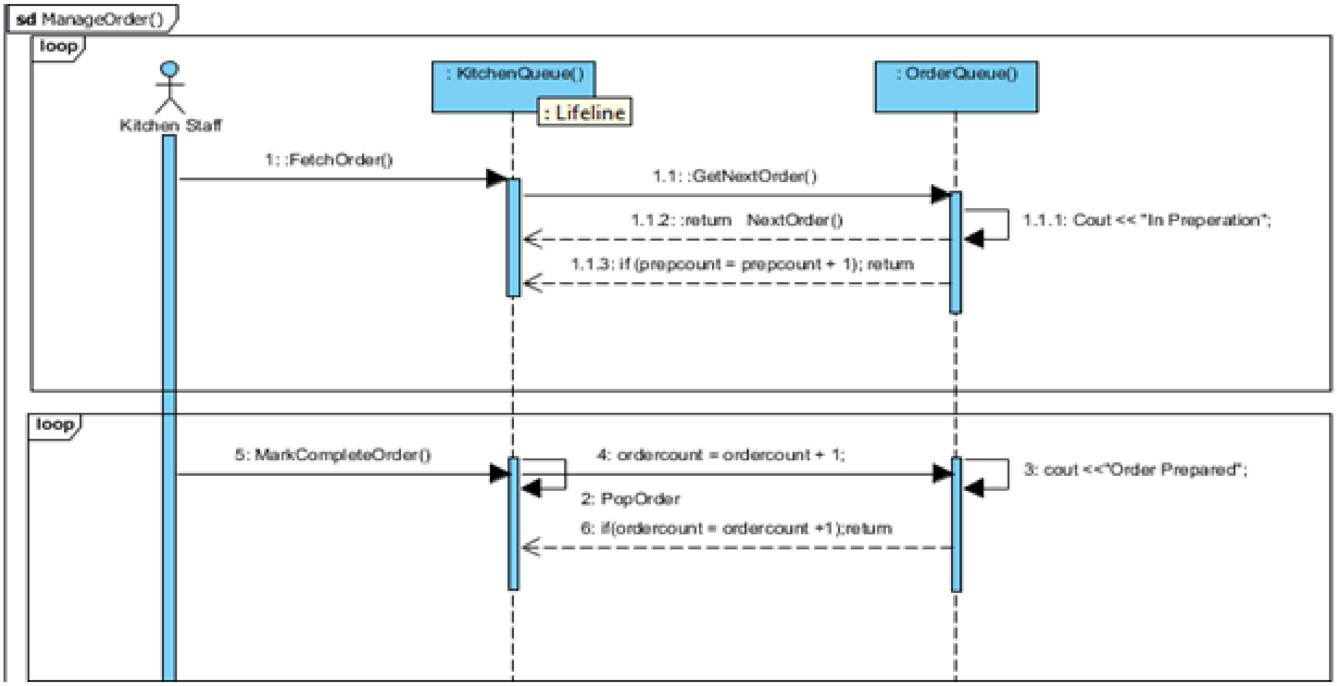
Here the manager can create a subdomain in GravyXpress for his/her restaurant. A manager sends a request to the Restaurant Builder class and this class asks for basic restaurant information such as address, working hours etc. The Manager sends this information to the Restaurant Builder who then creates the restaurant with this information. The Restaurant Builder also sends a request to the Customer Interface class to create an interface for the restaurant. The Manager is redirected to the Dashboard. One alternate plan is that the Manager class can request the Restaurant Builder class to add an employee by providing information about the employee. The Restaurant Builder will then request the Employee class to create such an employee and then it will add it to the restaurant. Another alternate plan is that the Manager class can request the Builder class to add a History section to interface for restaurant as shown above. The Manager class sends a request to the builder class who then sends a request to the Customer Interface class to add the details in the History part of the interface. One possible problem with this design is that the cohesion of the Restaurant Builder is slightly low since it has to respond to all of the requests of the Manager and then convey those requests to other classes such as Waiter and Customer Interface. In an alternate design, perhaps the Manager should be allowed to directly make a request to the Waiter and Customer Interface classes instead of having to reach them via Restaurant Builder. This would help to increase the cohesion of the Restaurant Builder. However, since the Restaurant Builder is an expert doer in managing the building of the restaurant, the current design also makes sense even though it has lower cohesion. Thus, there is always a tradeoff in designs.

ServeTable Use Case:



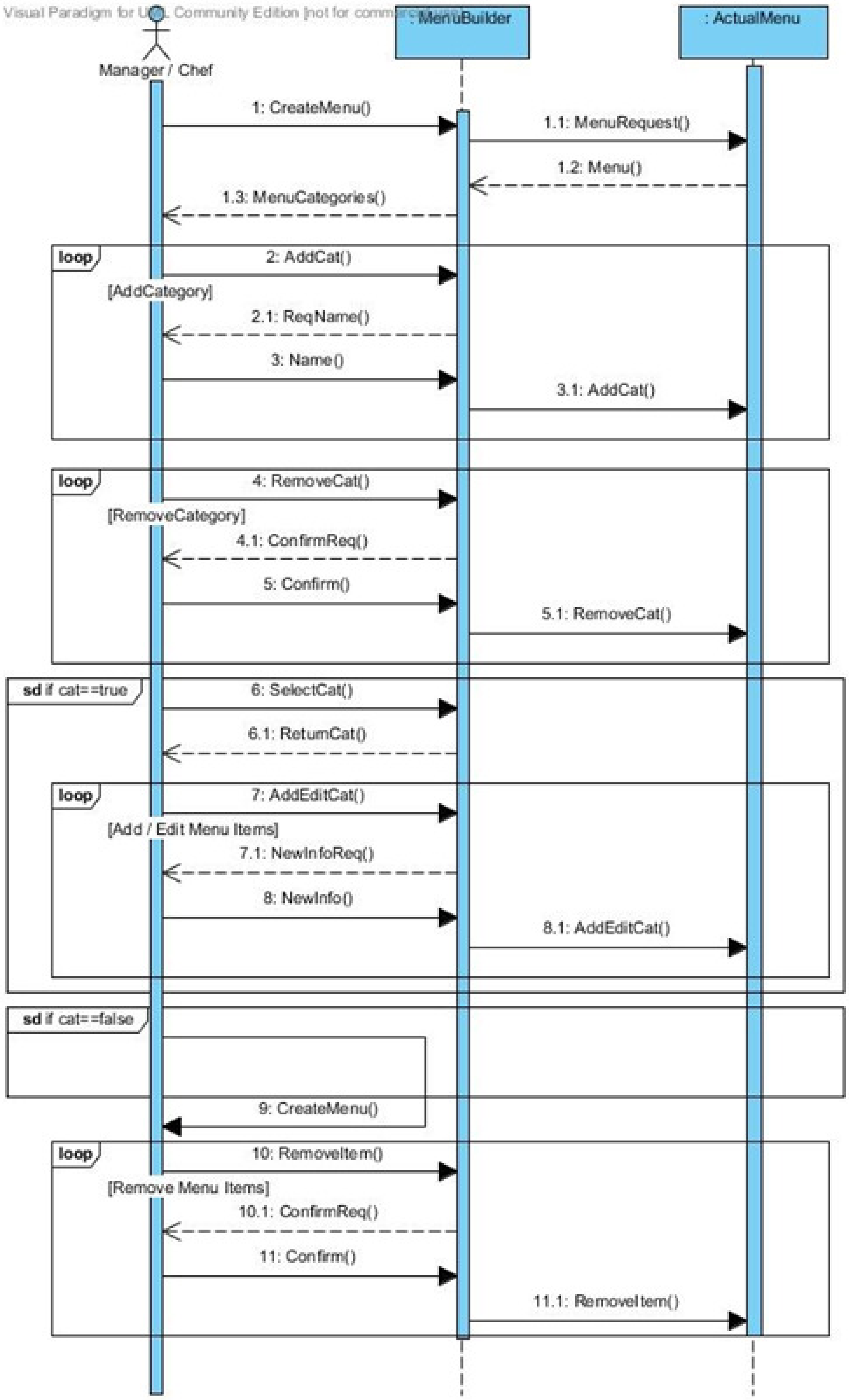
For a waiter to serve a table, he must login, using the login module. Once the login has been verified, he may request his assigned tables. A cookie will be set to ensure that he need not login again until he has closed his browser. He may then select a table from the list, and view the table's order. Upon delivery he may mark the order as served (only served orders will be reflected on the cheque.) He may perform a cash payment for the customer. Once the table has been cleaned, he may release it so additional customers may be seated.

ManageOrder Use Case:


sd ManageOrder()
loop
Kitchen Staff
: KitchenQueue()
: Lifeline
: OrderQueue()
1: :FetchOrder()
1.1: :GetNextOrder()
1.1.1: Cout << "In Preperation";
1.1.2: :return NextOrder()
1.1.3: if (prepcount = prepcount + 1); return
loop
5: MarkCompleteOrder()
4: ordercount = ordercount + 1;
3: cout <<"Order Prepared";
2: PopOrder
6: if(ordercount = ordercount +1);return

Here, a kitchen staffer manages orders. There are no log-ins required and the user's interaction is limited to clicking virtual buttons (order completed). The system then removes the order from the kitchen queue and notifies the customer, "Order Prepared". Alternatively, the system notifies the kitchen staffer and the customer when the order is "In Preperation". FetchOrder does as its name suggests and compiles an order from the queue. When an order is completed, MarkCompleteOrder acknowledges the order as complete and the systems returns to fetch the next order. KitchenQueue and OrderQueue indicate the queue of orders.

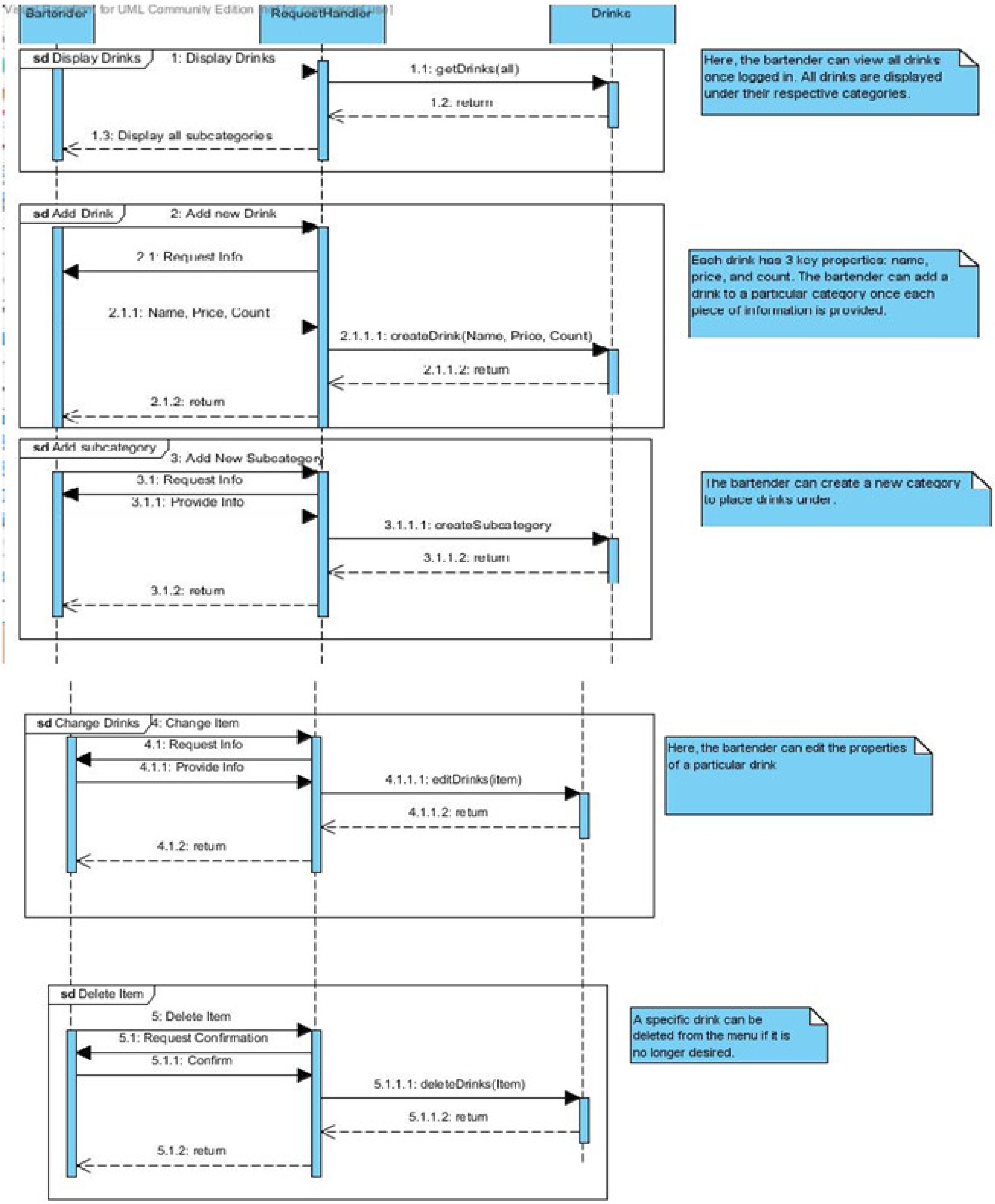
ChangeMenu Use Case:



Responsibilities:

- Manager or Chef (actor) begins to create menu for the restaurant.
- Menu builder sends request to ActualMenu which will get back to builder as Menu().
- Menu builder then requests to expand menu from actor by adding, removing or editing categories. For this task system will send MenuCategories() function to actor.
- Actor sends request for AddCat() which will return back to actor in form of RequestName() for category
- Actor then selects the Name() and it goes to actual menu after submission.
- Actor can also remove categories. Actor sends request RemoveCat() which will return back to actor from menu builder as ConfrimReq().
- Actor then confirms the request which removes the category from the actual menu.
- If item exists in the menu already "cat==true" check that and takes actor to edit the category, else "cat==false" takes actor back to CreatMenu() function where he can start over.
- Actor can also delete individual items. Where actor sends request to menu builder as RemoveItem() form which will then confirm the request from actor.
- Once actor confirmed the request it will then get deleted from the actual menu.

ChangeDrinks Use Case:



Here, the bartender can view all drinks once logged in. All drinks are displayed under their respective categories.

Each drink has 3 key properties: name, price, and count. The bartender can add a drink to a particular category once each piece of information is provided.

The bartender can create a new category to place drinks under.

Here, the bartender can edit the properties of a particular drink

A specific drink can be deleted from the menu if it is no longer desired.

2. Class Diagram and Interface Specification

A) Class Diagram:
The class diagram includes the following:

GravyXpress is a container class to hold all restaurant domains in the system. This also contains an about attribute representing generic data containing information about GravyXpress.

Restaurant is a class representing the structure of each overall restaurant. It contains attributes such as the restaurant's name, owner, unique identifier, and contact details. In addition, it contains a list of tables in the the restaurant, as well as the Restaurant and Kitchen Queues, and a list of the employees working at the restaurant.

Each of these compounded attributes is further broken down within its own class.

Employee is a class containing an Employee's data. A restaurant's Employees are aggregated into a main data structure where they are associated with their particular restaurant. The detailed Employee class is illustrated in the class diagram below.

Table is a class representing a physical table in a restaurant. The table has a unique identifier, in addition to the number of seats and its availability status.

Menu is a class with all the menu categories on a restaurant's menu.

MenuCategory is a class with the menu items in a category. The reason for separating menu into subsections is to allow for more flexibility, and the ability to perform operations on entire categories rather than merely individual menu items.

MenuItem is a class representing each item on the menu and the information associated with it.

OrderItem is a class that couples a menu item with a table and a timestamp to be ordered. The timestamp determines whether the order can be fetched by the kitchen.

OrderQueue is a data structure that collates these orders for the KitchenQueue and Cheque modules to access.

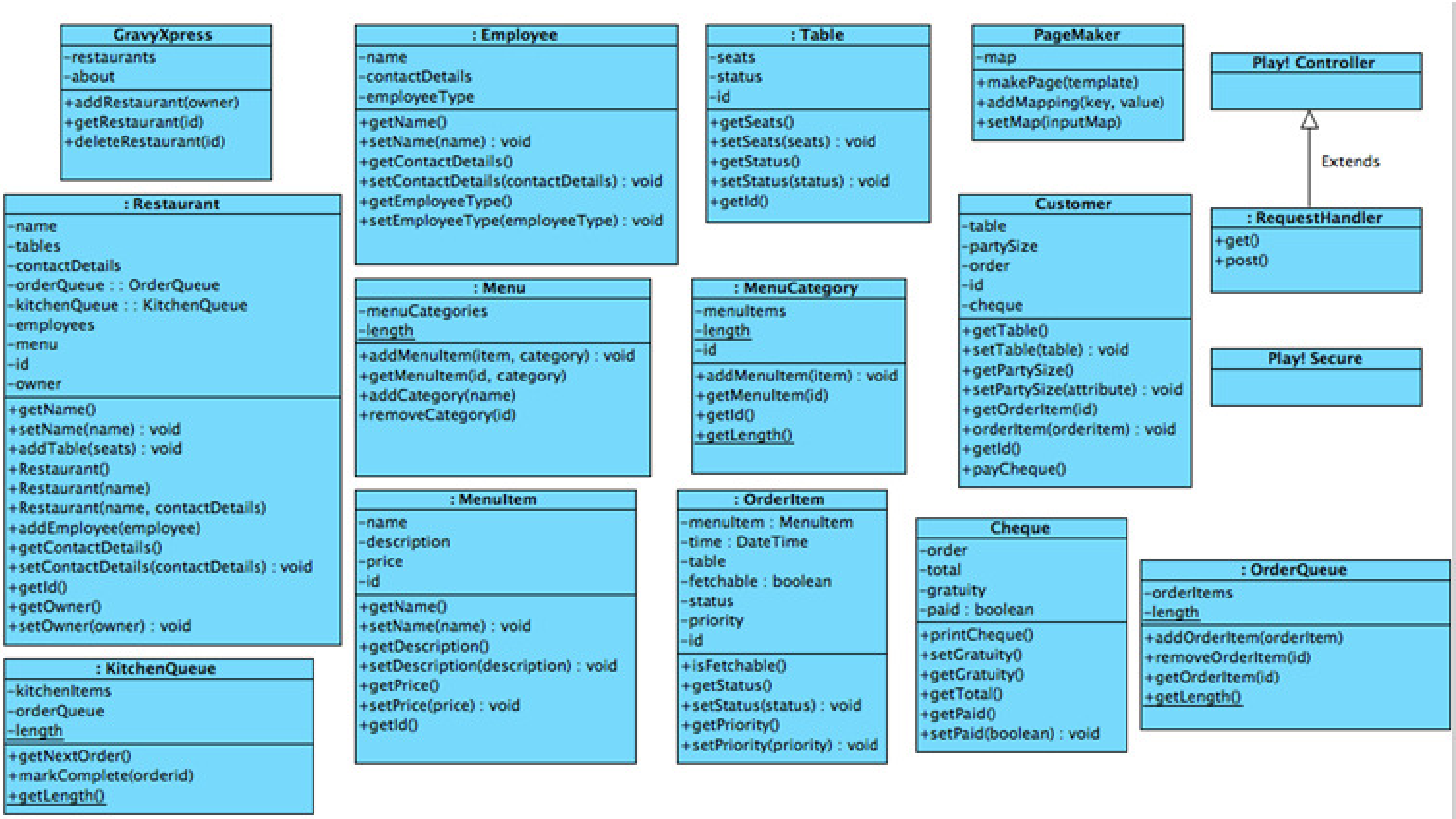
KitchenQueue is a class representing the queue that the kitchen sees. Items can be retrieved and updated from and within the OrderQueue.

Customer is a self explanatory class representing each customer, their party size, table location and cheque.

Cheque is an object used to store a dynamic cheque for a customer detailing his every purchase from the restaurant and enabling a gratuity feature.

Play! Controller is the controller class from the Play! webframework for Java. The **Request Handler** inherits from this class.

Play! Secure is Play!'s security module.



B) Data Types and Operation Signatures:

GravyXpress

Attributes:

-restaurants : string

// Name of the restaurant

-about : string

// Restaurant "about" description

Operations:

+addRestaurant(owner) : string

+getRestaurant(id) : string

+delRestaurant(id) : string

// adding/modifying restaurant's information

Restaurant

Attributes:

-name : string

-table : int

// Number of tables

-contactDetails : string

// Restaurant contact detail

-orderQueue : void

// Control over order queue

-kitchenQueue : void

// Control over kitchen queue

-employees : string

// Employees database

-menu : string

// Menu detail

-id : int

// Restaurant ID number

-owner : string

// Restaurant owner name

Operations:

+getName(): string

// Control over name

+setName(name) : void

+addTables(seats) : void

// Control over adding number of tables

+Restaurant() : string

+Restaurant(name) : string

// Control over restaurant name

+Restaurant(name, contactDetail) : string

// Control over restaurant contact detail

+addEmployee(employee) : string
// For adding employee's name
+getContactDetails(): string
// For employee's detail
+setContactDetails(contactDetails) : void
+getId() : int
// Control over restaurant's id
+getOwner(): string
// Control over owner name
+setOwner(owner) : void

KitchenQueue

Attributes:
-kitchenItmes : string
// Name of kitchen items
-OrderQueue : int
-lenght : int
// Length of orders

Operations:
+getNextOrder() : string
// Control on getting next order details
+markCompelete(orderId) : boolean
// If order is complete mark Yes, else No
+getLenght() : int

Employee

Attributes:
-name : string
// Name of the employees
-contactDetail : string
// Contact details of the employees
-employeeType : string
// Employees job title, such as waiter, chef etc

Operations:
+getName(): string
// Control over getting the name of the employee
+setName(name) : void
+getContactDetails(): string
// Control over getting the contact detail of the employee
+setContactDetails(contactDetails) : void
+getEmployeeType : string
// Control over getting employee's type
+setEmployeeType(employeeType) : void

Menu

Attributes:
-menuCategories : string

// Menu categories such as fastfood, breakfast etc
-length : int
Operations:
+addMenuItem(item, category) : void
+getMenuItem(id, category) : int
// Control on getting menu's ID
+addCategory(name) : string
// Adding the name of the menu's categories
+removeCategory(id) : boolean
// Control over deleting menu's categories

MenuItem

Attributes:
-name : string
// Menu item's name
-description : string
// Menu item's descriptions
-price : int
// Menu item's prices
-id : int
// Menu item's IDs
Operations:
+getName(): string
// Control over getting the name of the item
+setName(name) : void
+getDescription(): string
// Control over getting the description of the item
+setDescription(description) : void
+getPrice() : int
// Control over getting the price of the item
+setPrice(price) : void
+getId() : int
// Control over getting the id of the item

Table

Attributes:
-seats : int
// total seats associated with the tables
-status : boolean
// available / unavailable statuses
-id : int
// ID numbers of the seats
Operations:
+getSeats() : int
// Control over getting the seats
+setSeats(seats) : void

+getStatus() : boolean
// Control over getting the status of the seats
+setStatus(status) : void
getId() : int
// Control over getting the IDs of the seats

MenuCategory

Attributes:
-menuItems : string
-length : int
-id : int

Operations:
+addMenuItem(item) : void
+getMenuItem(id) : int
// Control over getting the menu item
+getId() : int
// Control over getting the menu's ID
+getLength() : int

OrderItem

Attributes:
-menuItem : string
// Name of the menu's item
-time : int
// Time for the order
-table : int
// Assigned table number
-fetchable : boolean
-status : boolean
-priority : string
// Priority such a high, medium & low
-id : int
// order ID

Operations:
+isFetchable() : boolean
+getStatus() : boolean
+setStatus(status) : void
+getPriority() : string
// Control over getting the priority of the order
+setPriority(priority) : void

PageMaker

Attributes:
-map : void

Operations:
+makePage(template) : void
+addMapping(key, value) : void

+setMap(inputMap) : void

Customer

Attributes:
- -table : int
  - // customer's desired table
- -partySize : int
  - // Number of person's for the order
- -order : int
- -id : int
  - // Customer's order id
- -cheque : double

Operations:
- +getTable() : int
  - // Control over getting the table
- +setTable(table) : void
- +getPartySize() : int
  - // Control over getting the party size
- +setPartySize(attribute) : void
- +getOrderItem(id) : int
  - // Control over getting the order items
- +orderItem(orderItem) : void
- +getId() : int
  - // Control over getting the order ID number
- +payCheque() : double
  - // Control over cheque

Cheque

Attributes:
- -order : int
- -total : int
  - // total price of the order
- -gratuity : int
  - // Calculated TIP for the order
- -paid : boolean
  - // Paid status

Operations:
- +printCheque() : boolean
- +setGratuity() : int
  - // Control over setting the gratuity
- +getGratuity() : void
- +getTotal() : int
  - // Control over getting the total price of the order
- +getPaid() : boolean
- +setPaid(boolean) : void

OrderQueue

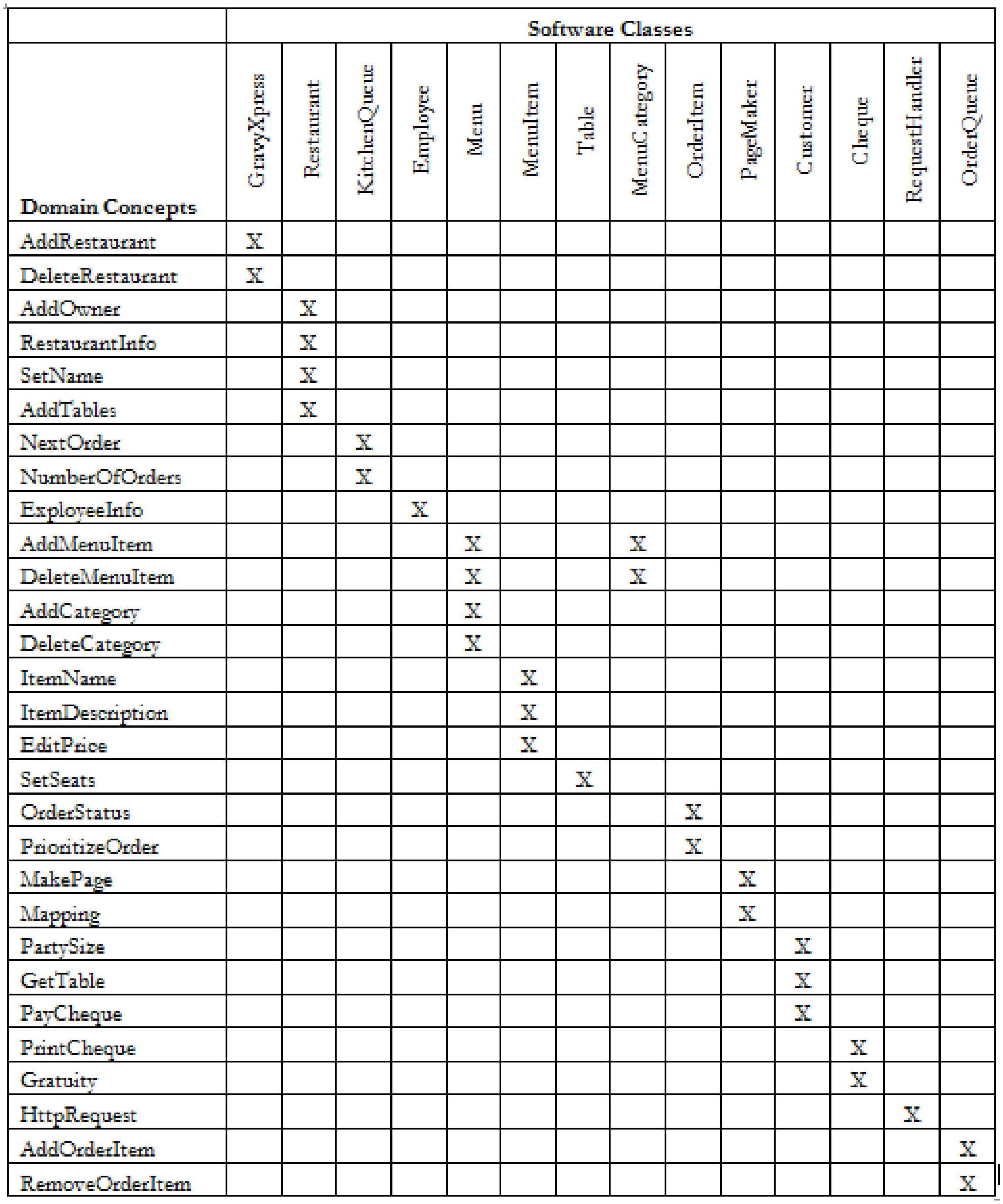
Attributes:
    -orderItems : string
    -lenght : int
Operations:
    +addOrderItem(orderItem) : void
    +removeOrderItem(id) : int
        // Control over removing the item from the cart
    +getOrderItem(id) : int
    +getLength() : int

C) Traceability Matrix:

Below is the traceability matrix that maps all software classes to all derived domain concepts. The development team realized that in the previous report, the original domain concepts were insufficient and not well developed. Therefore, more detailed domain concepts were derived when creating this traceability matrix. Most of the domain concepts were derived from what functions each class contained.

In addition, a RequestHandler class has been created in order to handle all webpage http requests. First, the user request will be processed through this class before being directed to the appropriate classes(s).

Two domain concepts (*AddMenuItem* and *DeleteMenuItem*) belong to two classes, *Menu* and *MenuCategory*. This is because adding/deleting a menu item or category will be done by the same member functions. The team will develop a virtual function where the system will decide which virtual function to use for the appropriate request.



	Software Classes													
Domain Concepts	GravyXpress	Restaurant	KitchenQueue	Employee	Menu	MenuItem	Table	MenuCategory	OrderItem	PageMaker	Customer	Cheque	RequestHandler	OrderQueue
AddRestaurant	X													
DeleteRestaurant	X													
AddOwner		X												
RestaurantInfo		X												
SetName		X												
AddTables		X												
NextOrder			X											
NumberOfOrders			X											
ExployeeInfo				X										
AddMenuItem					X			X						
DeleteMenuItem					X			X						
AddCategory					X									
DeleteCategory					X									
ItemName						X								
ItemDescription						X								
EditPrice						X								
SetSeats							X							
OrderStatus									X					
PrioritizeOrder									X					
MakePage										X				
Mapping										X				
PartySize											X			
GetTable											X			
PayCheque											X			
PrintCheque												X		
Gratuity												X		
HttpRequest													X	
AddOrderItem														X
RemoveOrderItem														X

3. System Architecture and System Design

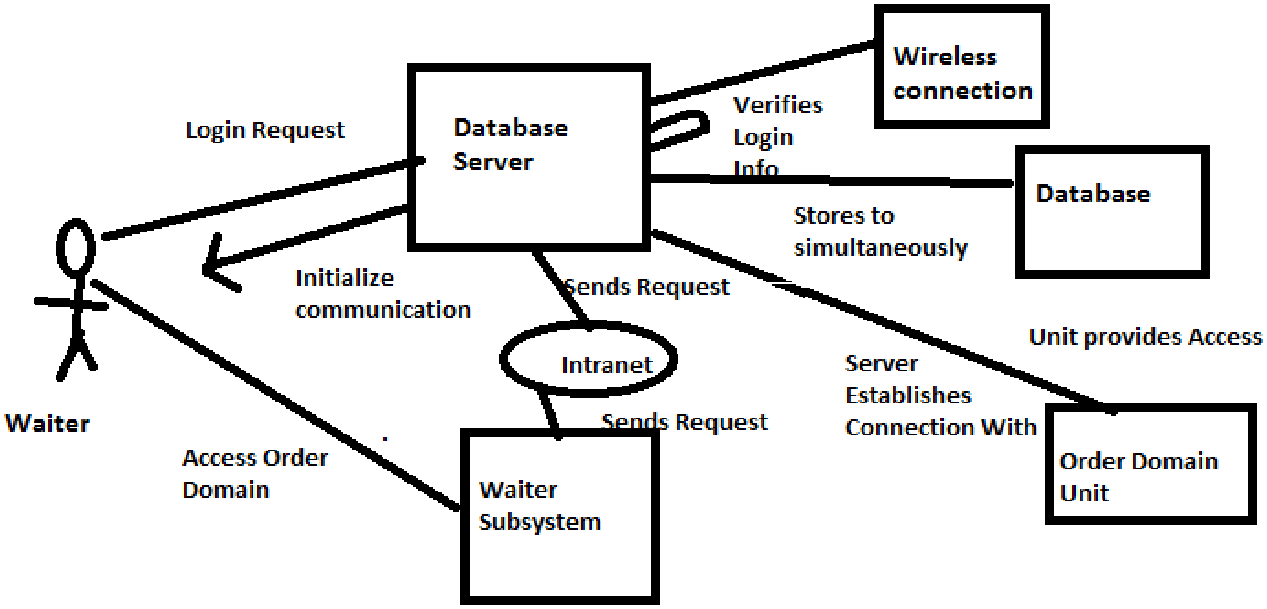
A) Architectural Styles:

The client-server architecture system we will use allows for multiple clients (such as managers, waiters etc.) to start communication sessions with and interact with the centralized database server. First, clients must successfully login to establish the connection. The clients can then connect to the services of the subdomains via the centralized server. This is a 2-tier architecture style since communication is directly between the client and the server.

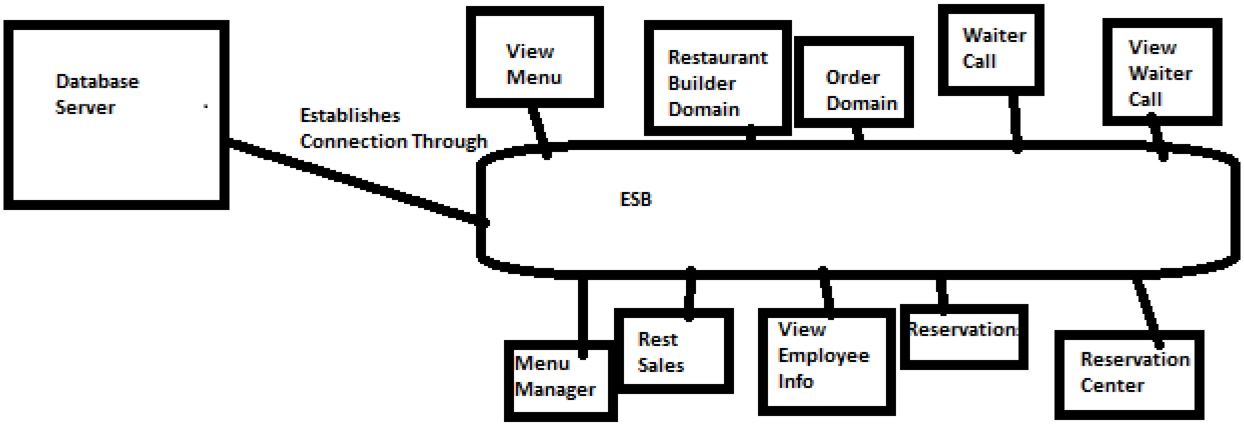
This client-server system is beneficial since it offers more centralized data (data stored only in server), has a better security(just need to control security of server), and is easier to maintain (roles are distributed among several subdomain units which connect via network). One downside of this system is its high dependence on the central server which can negatively influence system reliability.

Each user has his or her own terminal that uses a graphical UI to start communication with the database server. The server awaits requests from the clients and then passes along the requests to the correct subdomain. Clients can also communicate with one another through the server. The server also may read data and store in the database as needed. The database is used for backup purposes as well.

In the diagram below, the Client is the Waiter. After logging in with the Database Server, the Waiter is given access to communication with the server. When the Waiter requests to access the order domain unit, the Database allows for this by establishing a connection with the Order Domain Unit.



There will be one common bus to which the subdomain units connect to and this bus will be connected to the database server.The subdomain units and the server are connected via an Enterprise Service Bus. Thus, the database server can establish a connection to a particular subdomain unit via the ESB. When a request is sent from the client to the database server, it is passed along the ESB to the appropriate subdomain unit. The subdomain units are free to communicate with one another using the ESB and Subscribe/Publish message bus. Subdomains can use the subscribe/publish methods to read, write and update information by directly communicating amongst each other (without use of the server). This is quicker than going through the server and it also does not store the exchange of information in the database.

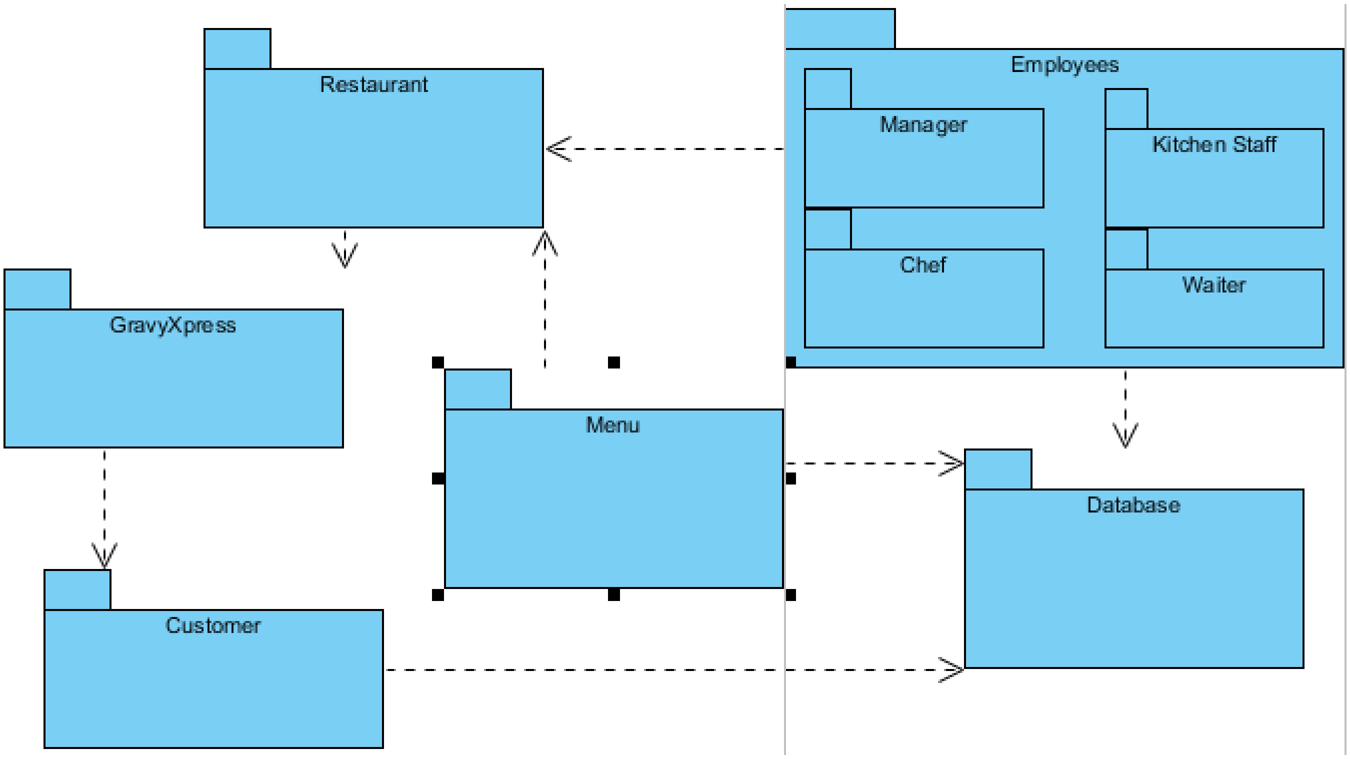


Note: There are other subdomains that were not shown due to lack of space in diagram

Overall, the client server and message bus system will define the overall architecture of the system. The client server model allows the clients to gain access to the subdomains and make requests to them. The message bus system allows for efficient communication amongst the server and the subdomains as well as amongst the subdomains themselves.

B) Identifying Subsystems and Package Diagram:

Package Diagram:



This maps out the main components of the software. Each individual package corresponds to several classes and use cases that have been documented prior to this. Each restaurant in GravyXpress will consist of its employees and its menu, the information for both of which will be stored in the database. Employees are of 4 main types: manager, chef, waiter, and the kitchen staff. Any customer of GravyXpress would use the web app to transact with the restaurant of their choice. Information about each customer is also stored in the main database.

Subsystems:

Each subsystem/set of subsystems must connect to the database server and fulfill the requirements indicated at the beginning of the report.

<u>List of Subsystems that Fulfill Requirements/User Stories:</u>

Restaurant Builder Domain - Used by Manager to create a restaurant subdomain within GravyXpress

Order Domain - Keeps track of Customers Orders, Customers can Add/Delete Items From Here

Kitchen Queue Domain - Keeps track of food items to be prepared in the Kitchen

Waiter Call and View Waiter Call Domains - Used for Customers to call upon Waiters and for Waiters to check which tables require their service (such as cleaning)

View Menu - Used by customers and employees to view the menu items

Menu Manager - Used by Manager, Chefs and Bartenders to edit the Menu

View Employee Info - Used by Manager to view the profiles of his/her employees

Employee Info - Used by Manager to edit Employee information/ Send messages to Employees

Restaurant Sales - Used by Manager to view restaurant sales

Reservation Center - Used by customers to make a reservation

Reservation - Reservation Center uses this subdomain to create a reservation

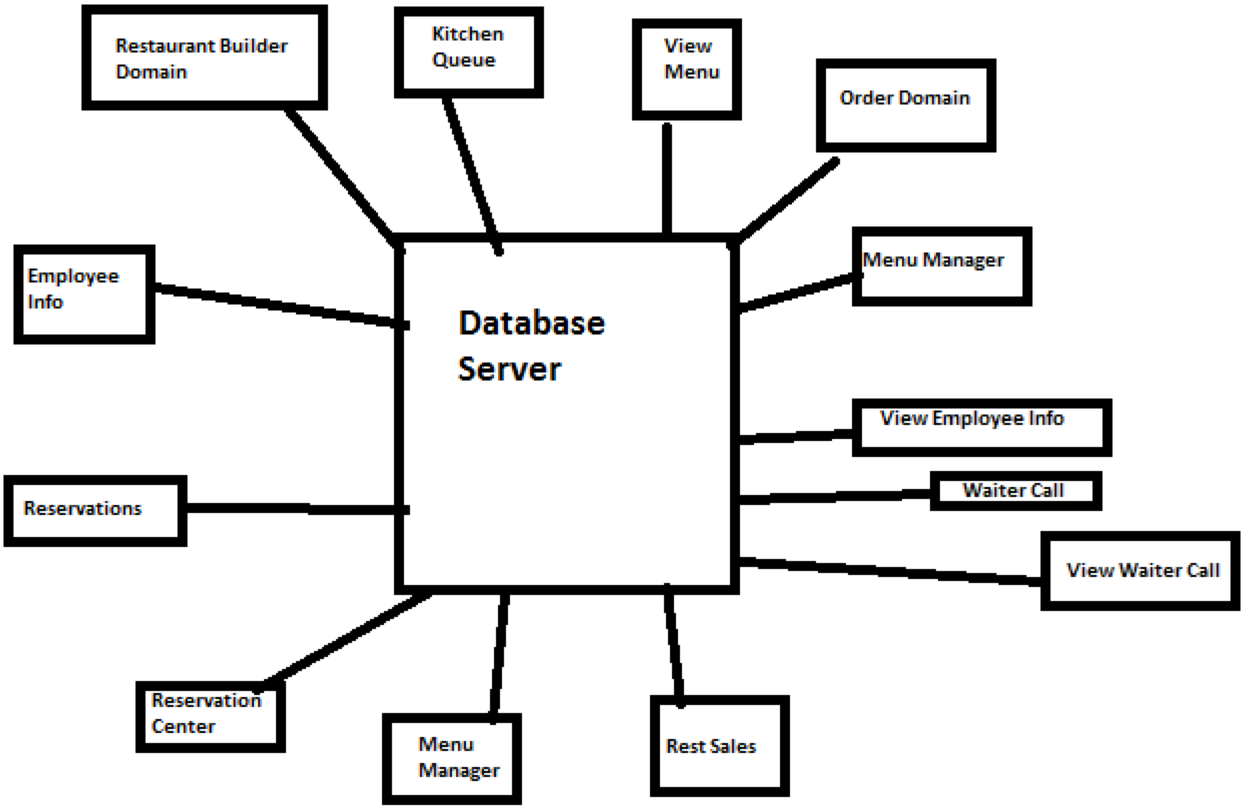
View Reservation - Used by employees to view the reservations

Payment - Used by customers to pay for the food they ordered

Interface Domain - Used for a customer interface and for a dashboard for the Manager to use

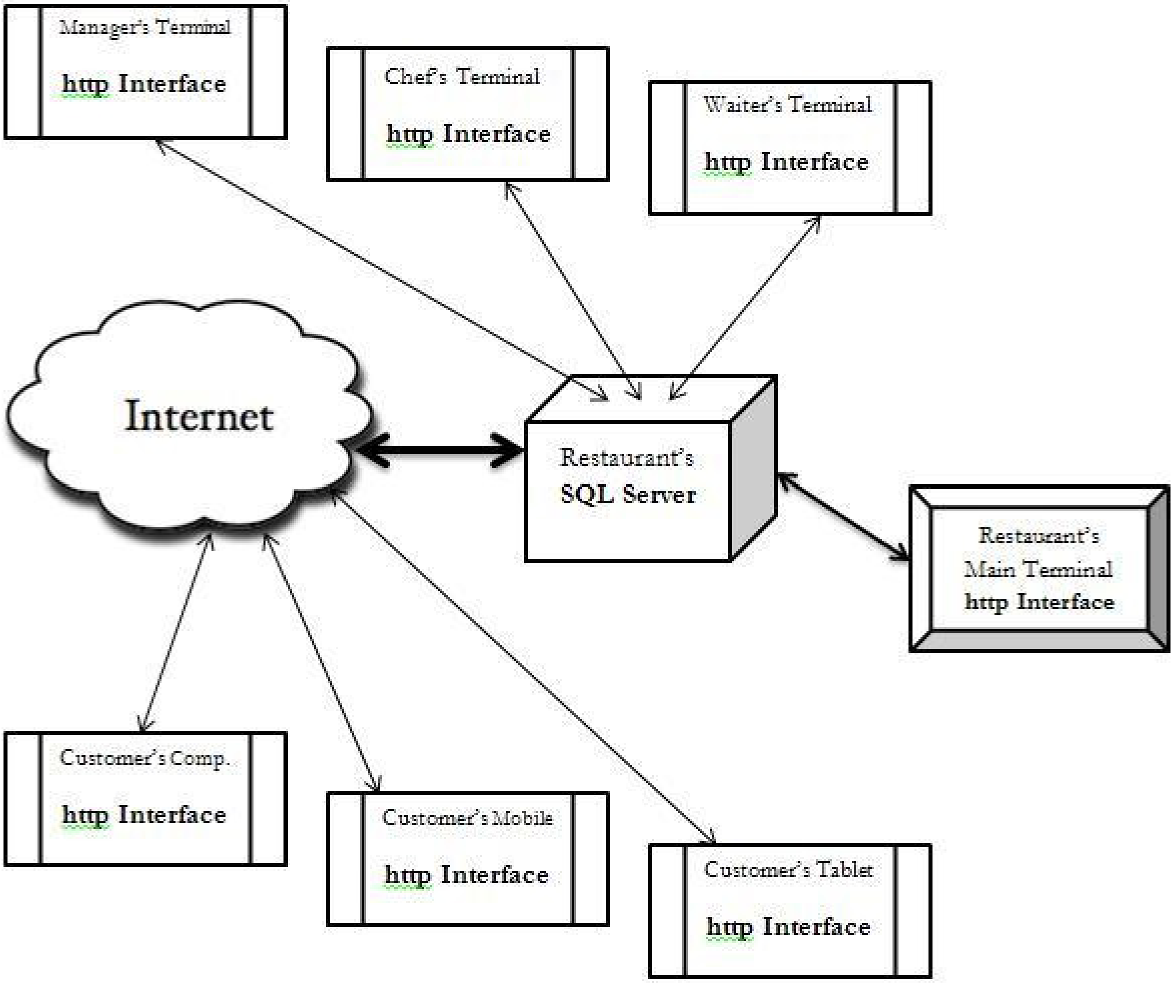
View Table Info - Used by employees to see which tables are available, dirty or clean

View Floor Layout/Floor Layout Domains - Used to see and update the floor layout of the restaurant

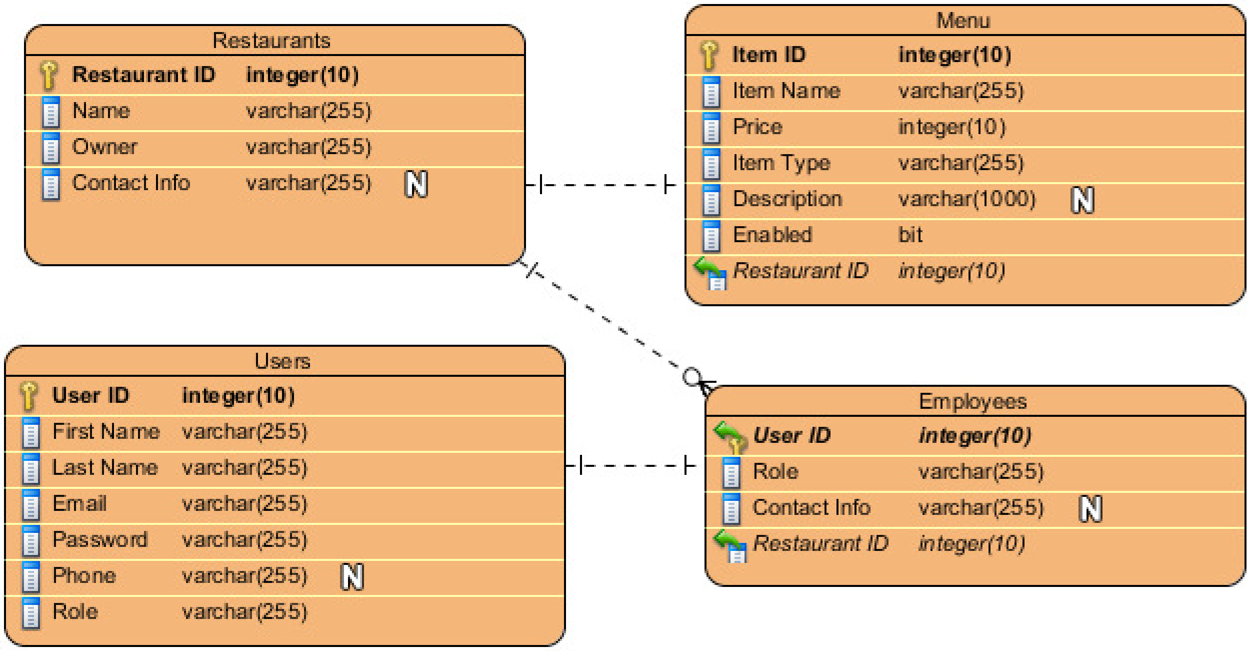


Note that all of these subdomains connect to the central database server, which is connected to the database. Thus, information can be sent and stored in the database as necessary. In the figure above, most of the subdomains are shown. There was not enough room to show all.

C) Mapping Subsystems to Hardware:



D) Persistent Data Storage:



Restaurants	
Restaurant ID	**integer(10)**
Name	varchar(255)
Owner	varchar(255)
Contact Info	varchar(255) N

Menu	
Item ID	**integer(10)**
Item Name	varchar(255)
Price	integer(10)
Item Type	varchar(255)
Description	varchar(1000) N
Enabled	bit
Restaurant ID	*integer(10)*

Users	
User ID	**integer(10)**
First Name	varchar(255)
Last Name	varchar(255)
Email	varchar(255)
Password	varchar(255)
Phone	varchar(255) N
Role	varchar(255)

Employees	
User ID	***integer(10)***
Role	varchar(255)
Contact Info	varchar(255) N
Restaurant ID	*integer(10)*

These 4 tables summarize the database design of GravyXpress. The users table is general to all users of GravyXpress, be they customer or employee of a restaurant. The user would use his/her e-mail as the username for their GravyXpress account. The employees table has a 1-to-1 relationship with the users table, and inherits the UserID as a foreign key. Each restaurant has one menu and several employees, thus the 1-to-1 and 1-to-many relationship with the menu and employees tables, respectively. Other objects, such as the OrderQueue and the customer's Cheque, are not stored in the database due to the fact that these objects are more dynamic in nature, and can easily be stored in main memory.

E) Network Protocols:

HTTP:

The main network protocol that GravyXpress will employ is HTTP. The choice of HTTP is an obvious choice given the webapp nature of GravyXpress. Browsers should be able to remotely deliver and retrieve data to and from the central GravyXpresss server. Such requests map easily to the GET and POST requests native to HTTP.

HTML5 Websockets

In addition to HTTP, we will be utilizing the fairly new websocket protocol. The websocket protocol is advantageous in its capability to enable servers to send content to clients that has not been solicited by the client. This is achieved by keeping a connection opened by the client open, and passing data along this channel back to the client.

Modern browsers support HTML5 websockets, and for real-time alerts this protocol is far superior to a constant barrage of HTTP requests sent by the client to solicit content, such as is achieved with Comet or other similar technologies. It is also simpler.

Our application will use websockets to push updates to users in real-time.

F) Global Control Flow:

Execution orderness: GravyXpress is both a procedure-driven and an event-driven software. When the customer orders food, manager or chef adds or removes an item from the menu, or the manager creates a subdomain for his particular restaurant, the user(s) must all go through the same steps every time for each goal. For example, in the process of ordering food, the restaurant customer must select the "View Menu" option. Then, he/she must choose a food category (Pizza, Pasta, Sandwiches, Drinks, etc.). Then, the customer must choose the food item he/she desires from the list in that category. Finally, the customer must select the order button to send to the order queue. This is all procedure-driven, or in other words, the customer must always go through these steps in order to complete the ordering process. There are many more procedure-driven events that cannot all be described here.

GravyXpress is also event-driven in that it stays idle in a loop until an action is taken while the user tries to accomplish his/her goal. For example, the system is in fact running and already in a loop when the restaurant customer sees the main menu of the system in the tablet or smartphone. Another example is when the restaurant customer selects a menu item in a particular category. When the customer selects the category, the system goes into a loop (idle) until the customer then selects the menu item he/she wants to order.

Time dependency: GravyXpress has multiple timers. The system will have a timer when the user starts a procedure to accomplish his or her goal. A timer will start and reset every time an action is taken during the procedure. If the user does not take any action and the timer reaches a maximum allowed time, a "time-out" will occur where the system will give out a "Time-out" message and will go back to the beginning of the procedure and reset the timer. So for example, if the restaurant customer does not do anything for a long time after selecting a menu category, the system will go back to its Main Menu.

Another timer that will be used is for the current date and time. When the customer orders food and requests the cheque, the system records the date and time the customer ordered food and prints the date and time on the cheque. This will also be sent out to the manager who also wants to see the date and time each customer ordered food.

The timer that starts and resets between procedures is not considered real-time since it keeps resetting and there is a maximum threshold where it will reset automatically if no user action is taken. However, the timer used to display the date and time a customer ordered food is considered real-time, since it uses the actual date and time outside the system.

Concurrency: GravyXpress will be processing multiple requests at the same time. Customers will be ordering food at the same time as the manager will be viewing the order history. The kitchen staff will be marking orders complete as orders will simultaneously move from the order queue to the kitchen queue. Many other concurrent processes will occur that are too great in numbers to describe fully here. As a result, multiple threads will be used for multiple processes.

G) Hardware Requirements:

SQL Server

Our service will use an SQL database to store orders, ingredients, menu items, etc. We plan to use a server with the following minimum and desired performance requirements. These are the 2012 edition specifications for the SQL server. Some restaurants will need a server with demanding performance requirements, so the Recommended Requirements will be set as our default specification setting.

Hardware Component	Minimum Requirements	Recommended Requirements
Processor	1.0 GHz	1.4 GHz
RAM	512 MB	1.0 GB
Hard Drive Space	3.6 GB	4.0 GB
Network	10/100/1000 NIC Wifi 802.11n	10/100/1000 NIC Wifi 802.11n

Desktop Client

Many restaurants have desktop terminals that employees will interact with, so the following table provides minimum and recommended requirements for a desktop client. The most common desktop monitors are between 19 and 20 inches, so recommended requirements will be set as our default specification setting.

Hardware Component	Minimum Requirements	Recommended Requirements
Processor	1.0 GHz	1.4 GHz
RAM	512 MB	1.0 GB
Hard Drive Space	4.2 GB	6.6 GB
Network	10/100/1000 NIC Wifi 802.11n	10/100/1000 NIC Wifi 802.11n
Screen Size	15”	17”-19” or 20”
Resolution	1024 x 768	1280 x 1024 or 1600 x 1200

4. Algorithms and Data Structures

A) Algorithms

There are many important algorithms that will help us to implement the use cases. Most of our algorithms are not complex in nature. For instance, many algorithms will deal with adding items to and deleting items from linked lists. Since we will be using lists to represent the OrderQueue and the KitchenQueue, whenever the waiters or chefs must update and alter these queues such algorithms will come in handy.

Furthermore, another algorithm that can be used to calculate total bills is the algorithm for summing all the terms in an array. If each item ordered is placed in an array cell along with its price, one can see how this algorithm will come in handy at bill time.

Some search/sort algorithms which are more complicated will also be used as well. Sometimes data records about sales need to searched. Other times, to figure out the popularity of items, items must be sorted according to the largest number of sales. Such information helps the manager run the restaurant.

An example algorithm to check the stock of restaurant items, checkStock, can retrieve the amount of each item in the restaurant's inventory. An added function can be used to display a bar graph displaying the amount of data. A smaller algorithm that can be a part of checkStock can also sort items into different categories to make it easier for the manager to view data. With this algorithm the data can be further divided into Alcohol/Wine brands and quantities, spices (salt, pepper, oregano, etc) quantities, types of grains (wheat, flour, etc), and even foods that can be possible allergens (shellfish, peanuts, etc). This shows how algorithms are an essential part of any software system and how they are so beneficial.

Overall, Algorithms are a very crucial part of GravyXpress. They help facilitate the implementation of all key user stories in an efficient manner.

B) Data Structures

There are several key data structures that we will use in this project, for instance, arrays, linked lists, lists, priority lists, queues, stacks etc. Data structures are very vital to the efficiency of the software system. When it comes to improving upon the time it takes for execution of algorithms and improving on performance, arrays will be used. Arrays are by far the fastests from a performance point of view. For instance, Arrays can be used to store a customer's order. Each item that is ordered can be placed in a cell of the array and then then an algorithm for finding the sum of all elements in the array can be used to to calculate the price the customer should pay.

However, for implementing certain aspects of our project it makes the most sense to use lists. For example, in order to implement the "Kitchen Queue" we will use a priority list. This list will work much like a queue in the sense that items that are entered in the queue first will be the first to be cooked. Thus, items that are entered first are given priority over items that are entered last. The priority list will also have several pointers that we will use to implement the functions of the "Kitchen Queue." For instance, there will be a pointer at the beginning and end of the list. There will also be a pointer called "CooksHere." To the left of the

“CooksHere” pointer will be food items that have already been assigned to a chef to cook. To the right of the “CooksHere” pointer will be food items that have not been cooked already. Also, there will be another pointer called “ReadyToServe” which will indicate which items have already been cooked and are ready to serve and which are not. When an item is ready to serve, the Waiter for the table it is to be sent to will be notified. To the left of the “ReadyToServe” pointer will be food items that are ready to serve and have been cooked, to the right will be items that are not ready to serve.

Another instance where a list will be used will be for OrderQueue. The list will also be a priority list since items entered in the list first will be the items that exit the list first. There will be a pointer called “SentToKitchen” which will distinguish between the items sent to the kitchen queue and those that have not been sent to the kitchen queue. To the left of the pointer will be items sent to the KitchenQueue already and to the right will be items that have yet to be sent. Another pointer that will be used will be the “Delivered” pointer. To the left of this pointer will be items that have already been sent to kitchen, have been cooked, and have been sent by the waiters to the customers. The items to the left of the “Delivered” pointer will be added to the cheque for the corresponding customers. The items to the right will be those that have not been delivered to the customers yet by the waiters.

Last but not least, hash tables can be used to map keys to values. This can be used for many aspects of implementation. For instance, a hash table can be used to map the keys (employee names) to their values (payroll amount, telephone number etc.).

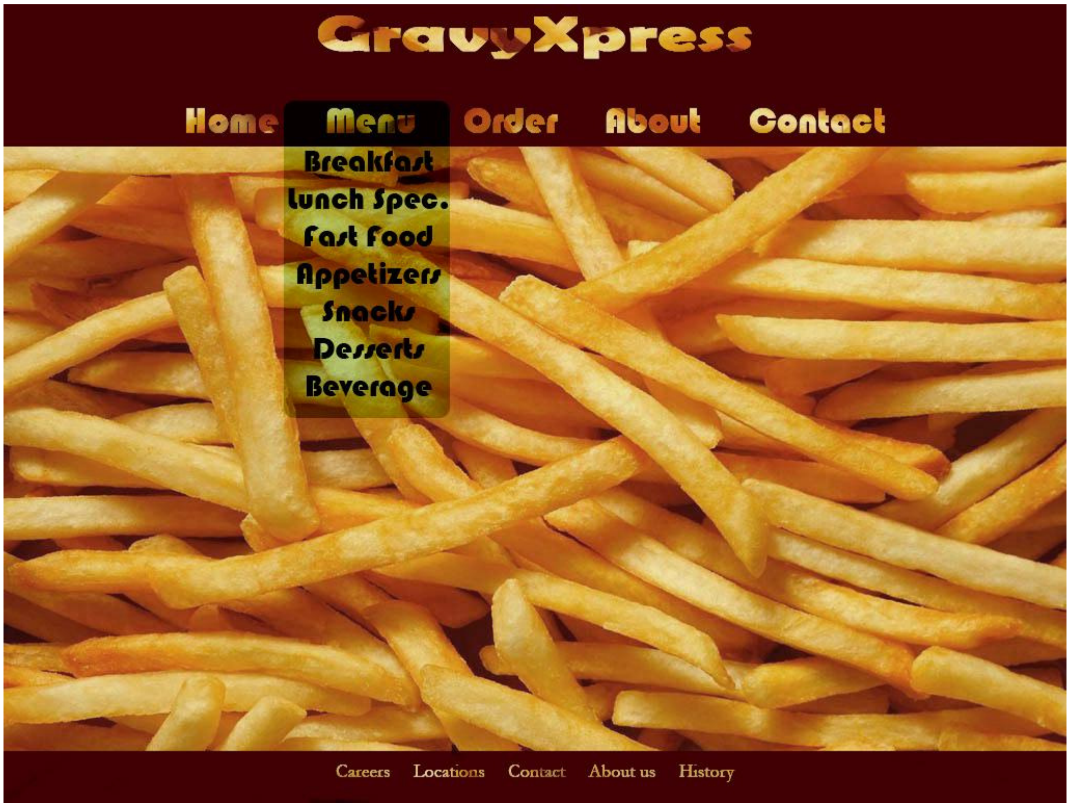
5. User Interface Design and Implementation

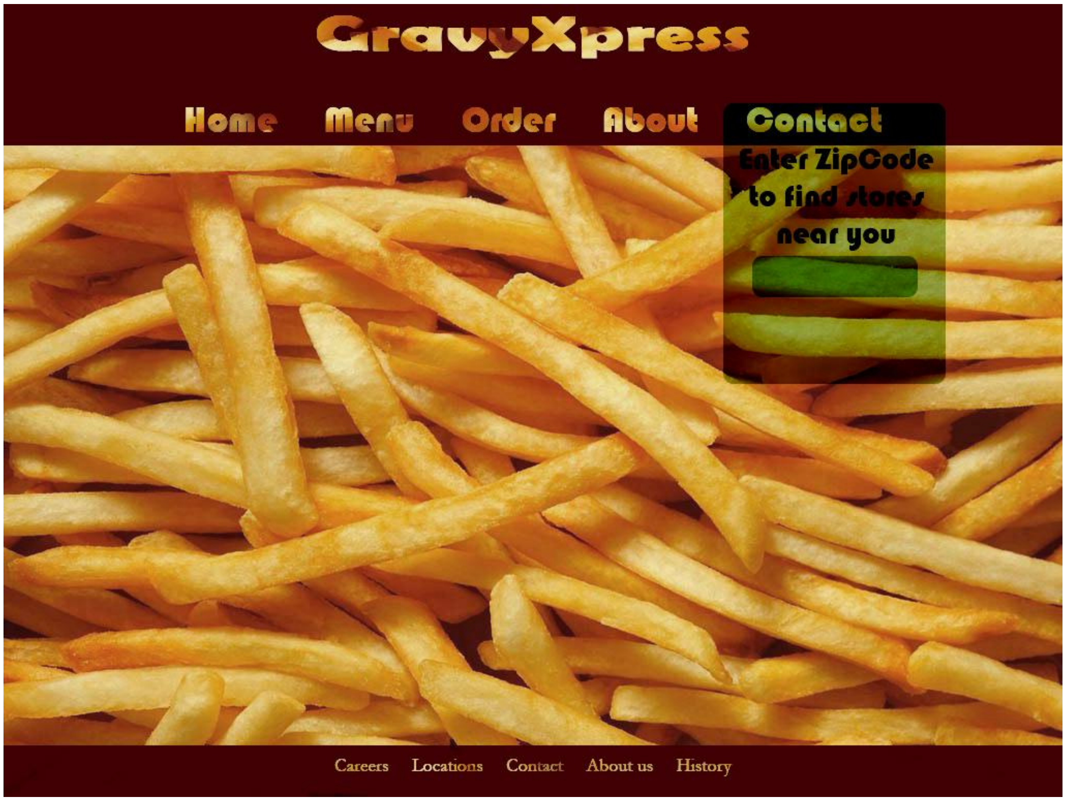
On the next page is our improved graphical user interface design in which we focused mainly on ease-of-use rather changing the colors and design as described in the description of report.Customers can use their own mobile device or a tablet to access the GravyXpress app. After sitting down on a table, the waiter will give the customers at the table he/she is serving the tablet showing the Home page shown below. Here Customers may select to view different sections of the menu (as shown by the list below) and may order food, view the order and pay the bill. Under the menu tab show below, customers can view the different menu items broken up by category. For instance, there is a breakfast, lunch special, fast food etc. category. Improved user interface will have hover over menus. That is, if user will hover the mouse pointer or on touch interface user's first touch to menu buttons will give him drop down menu which will give him sneak peek of what's inside. These menus will be implemented using a mixture of CSS and Javascript. The CSS will be used to style the HTML elements, while Javascript will be used to alter the HTML DOM as needed. All of this will be handled on the client side so no requests need to be made to the server in order to achieve this functionality. This maximizes speed, and minimizes the complexity of the code.

Furthermore, under the order page, customers can select menu items to order and then place their order. The prices of the items are shown on the order as well. The Contact tab will have an active hover over drop down menu. That is, it will give user a text field in which user can input the zip code and hit enter. It will then give him the nearest stores information. The About tab will provide general information about the Restaurant such as hours of operation and special offers on certain days. The About tab will also have some information about the hotel manager. The design of the Main Portal (in the previous report) is still the same. It can be used by the Manager to create a subsystem for his/her Restaurant under GravyXpress.

Towards the bottom of the home page, there are also links for Careers, Locations, Contact, About Us and History. Careers is where the manager may wish to post job openings for the public to see and apply to. Locations will list the location(s) of the restaurant. The contact link is the site contact. The History link provides the user with the history and heritage of the restaurant. This adds a cultural aspect to the design.

Although these pages like all other pages will be served dynamically, these pages themselves will be served from static HTML styled with CSS styling sheet content. These pages merely serve content to the end-user and no content needs to be sent from the client to the server, making these pages very simple to render.


GravyXpress
Home
Menu
Order
About
Contact
Breakfast
Lunch Spec.
Fast Food
Appetizers
Snacks
Desserts
Beverage
Careers
Locations
Contact
About us
History


GravyXpress
Home
Menu
Order
About
Contact
Enter ZipCode
to find stores
near you
Careers
Locations
Contact
About us
History

6. Design of Tests

A) Test Cases

OrderFood

Test-case Identifier:	TC-1
Use Case Tested:	UC-1, main success scenario
Pass/Fail Criteria:	System passes test if the restaurant customer successfully places order, and at the end of the process the system tells the customer that "Food is Ready and Arriving" and goes back to main menu after 1 minute.
Input Data:	Food Item

Test Procedure:	**Expected Result:**
Step 1. Select "View Menu" option.	System displays menu categories (drinks, appetizers, specials, lunch, dinner, etc.)
Step 2. Select any food category.	System displays all items in that category, price of the item, attach note option, and the "Add to Cart" option.
Step 3: Choose any food item and select the "Add to Cart" option next to that item.	System keeps a count of the number of items and calculates the total cost, both at the bottom-right corner of the screen. System automatically sends the orders from the cart to the kitchen's order queue.
Step 4: Select the "View Status" option.	System displays the status of order (whether it's "In Order Queue" or "In Kitchen Queue") and has the options to "Add More Items" or "Remove Items". When system displays "Food is Ready and Arriving", system goes back to main menu after 1 minute.

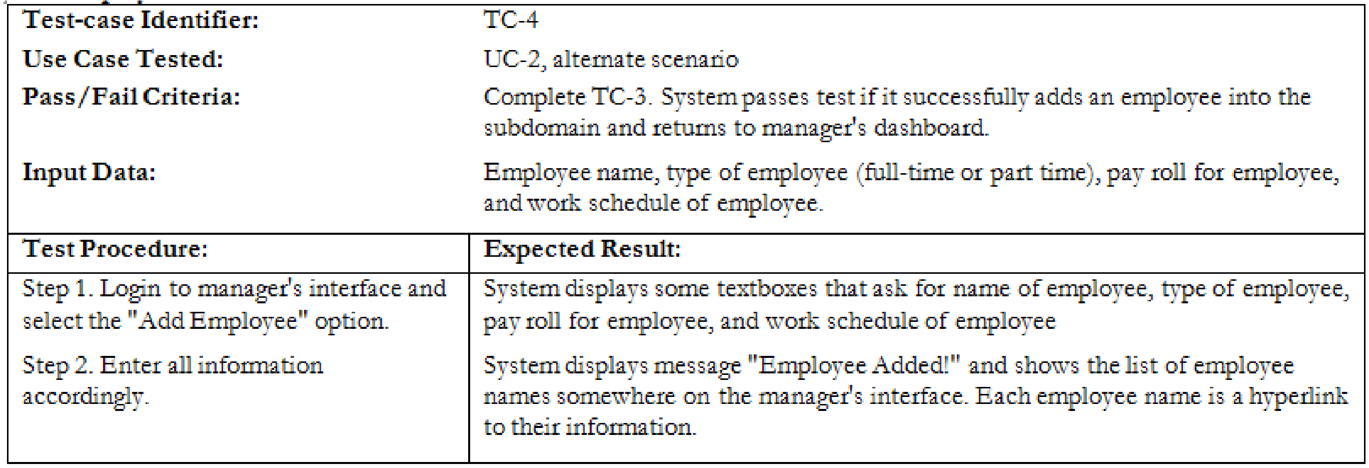
RemoveOrder

Test-case Identifier:	TC-2
Use Case Tested:	UC-1, alternate scenario
Pass/Fail Criteria:	Complete TC-1 first. System passes test if it removes ordered items while the status reads "In Order Queue", and displays error message if status reads "In Kitchen Queue".
Input Data:	"Remove Items" option is selected.
Test Procedure:	**Expected Result:**
Step 1. Select "Remove Items" option.	System displays list of all ordered items, their statuses ("In Order Queue" or "In Kitchen Queue"), a check box next to each item, and a "Delete Item" button at the bottom of screen.
Step 2. Check any item whose status reads "In Order Queue".	System checks the check box of that item and also highlights the entire row.
Step 3: Select the "Delete Item" button at the bottom of screen.	System successfully removes the item from the list.
Step 4: Check any item whose status reads "In Kitchen Queue"	System checks the check box of that item and also highlights the entire row.
Step 5: Select the "Delete Item" button at the bottom of screen.	System displays an error message "Item being prepared. Cannot delete item!" System returns to the "Remove Items" list.

CreateWebpage

Test-case Identifier:	TC-3
Use Case Tested:	UC-2, main success scenario
Pass/Fail Criteria:	System passes test if it displays a subdomain with the restaurant manager's preferences.
Input Data:	Restaurant name, manager name, hours of operation, address
Test Procedure:	**Expected Result:**
Step 1. Go to GravyXpress application and select "Create My Restaurant" option.	System displays some textboxes that ask for name of restaurant, manager name, hours of operation, and address.
Step 2. Enter all information accordingly.	System displays the restaurant name, manager name, hours of operation, and address on the home page of the user interface for the restaurant. System provides manager with a dashboard interface.

AddEmployee



Test-case Identifier:	TC-4
Use Case Tested:	UC-2, alternate scenario
Pass/Fail Criteria:	Complete TC-3. System passes test if it successfully adds an employee into the subdomain and returns to manager's dashboard.
Input Data:	Employee name, type of employee (full-time or part time), pay roll for employee, and work schedule of employee.
Test Procedure:	**Expected Result:**
Step 1. Login to manager's interface and select the "Add Employee" option.	System displays some textboxes that ask for name of employee, type of employee, pay roll for employee, and work schedule of employee
Step 2. Enter all information accordingly.	System displays message "Employee Added!" and shows the list of employee names somewhere on the manager's interface. Each employee name is a hyperlink to their information.

ServeTable

Test-case Identifier:	TC-5
Use Case Tested:	UC-3, main success scenario
Pass/Fail Criteria:	System passes test if waiter is informed of table's order status and whether customer has left at the end.
Input Data:	All of the assigned table's orders.
Test Procedure:	**Expected Result:**
Step 1. Login to interface and select the "Assigned Tables" option.	System displays the table numbers that the user is to serve, the status of the table's orders (Ordering, In Order Queue, In Kitchen Queue, Order Ready, Served), and the table's cheque hyperlink.
Step 2. Select one of the table numbers assigned.	System displays the order details of the table, including table number, the price of each item, and order status. Order status becomes "Order Ready" for that table number. The entire table row is highlighted in green.
Step 3. Select the "Acknowledged" button.	System changes the status of table to "Served" and removes the highlighting. System alerts user that table wants to pay by cash. Displays "Payment by Cash" for that table and highlights the row yellow.
Step 4. Select the "Acknowledged" button.	System removes the "Payment by Cash" message and removes the highlighting. System alerts user that table needs to be cleaned. Displays "Cleaning Required".
Step 5. Select the "Acknowledged" button.	System deletes the table from the list of tables to serve.

CustomerAssistance

Test-case Identifier:	TC-6
Use Case Tested:	UC-3, alternate scenario
Pass/Fail Criteria:	System passes test if waiter is successfully signaled to assist restaurant customers.
Input Data:	Call to assistance from waiter.
Test Procedure:	**Expected Result:**
Step 1. Login to interface and select the "Assigned Tables" option.	System displays the table numbers that the user is to serve, the status of the table's orders (Ordering, In Order Queue, In Kitchen Queue, Order Ready, Served), and the table's cheque hyperlink.
Step 2. At any moment of time, a table signals for waiter's assistance through system.	System displays a pop-up message that informs waiter "Assistance Required. Table: X" where X is the table number. System also highlights that table row as red.
Step 3. Select the "Acknowledged" button.	System closes the "Assistance Required" message and removes highlighting.

ManageOrder

Test-case Identifier:	TC-7
Use Case Tested:	UC-4, main success scenario
Pass/Fail Criteria:	System passes test if it can show the list of orders in the order queue, have option to move orders to kitchen queue, and have option to view notes on some orders.
Input Data:	List of incoming orders in the order queue.
Test Procedure:	**Expected Result:**
Step 1. In the shared kitchen staff interface, select "View Order Queue".	System displays the list of orders in the order queue.
Step 2. Select the "View Kitchen Queue" option.	System displays the list of orders in the kitchen queue.
Step 3. In the order queue, select any number of orders using check boxes and select the "Move to Kitchen Queue" option.	System removes the order from the order queue. System adds the same order into the kitchen queue.
Step 4. Select order completed button next to completed order item.	System removes the item from the kitchen queue. In the waiter's interface, system displays "Order Ready" message for that order. In the customer's interface, system displays "Food is Ready and Arriving" message for that order.

CancelOrder

Test-case Identifier:	TC-8
Use Case Tested:	UC-4, alternate scenario
Pass/Fail Criteria:	System passes if the order queue refreshes and removes any orders that were cancelled by customer. The kitchen queue is not affected.
Input Data:	Command to cancel order.
Test Procedure:	**Expected Result:**
Step 1. In the shared kitchen staff interface, select "View Order Queue".	System displays the list of orders in the order queue.
Step 2. Customer interface sends a message to cancel an order that is already in the order queue.	System deletes the order from the order queue.
Step 3. Customer interface sends message to cancel an order that is in the kitchen queue.	System does NOT delete the order in the kitchen queue.

ChangeMenu

Test-case Identifier:	TC-9
Use Case Tested:	UC-5, main success scenario
Pass/Fail Criteria:	System passes if the chef can successfully change the restaurant menu by adding an item into a category.
Input Data:	Name of item, price, and inventory count.
Test Procedure:	**Expected Result:**
Step 1. Login to chef's interface and select the "Create Restaurant Menu" option.	System displays existing menu categories (appetizers, lunch, specials, etc) (if any) and gives the option to "Delete" next to each category. At the beginning of list is the "Add New Category" option. Suppose categories do exist.
Step 2. Select a category to add a new food item.	System displays names of all items in that category including price, inventory count, and gives options to "Delete Item" and "Change Item" for each item. At beginning of list, system gives option to "Add New Item".
Step 3. Select the "Add New Item" option.	System displays pop-up window with some textboxes that ask for the "Name:", "Price:", and "Inventory Count:" of the new item.
Step 4. Enter information and hit the "Add Item" button when finished.	System adds the item into the list in alphabetical order.

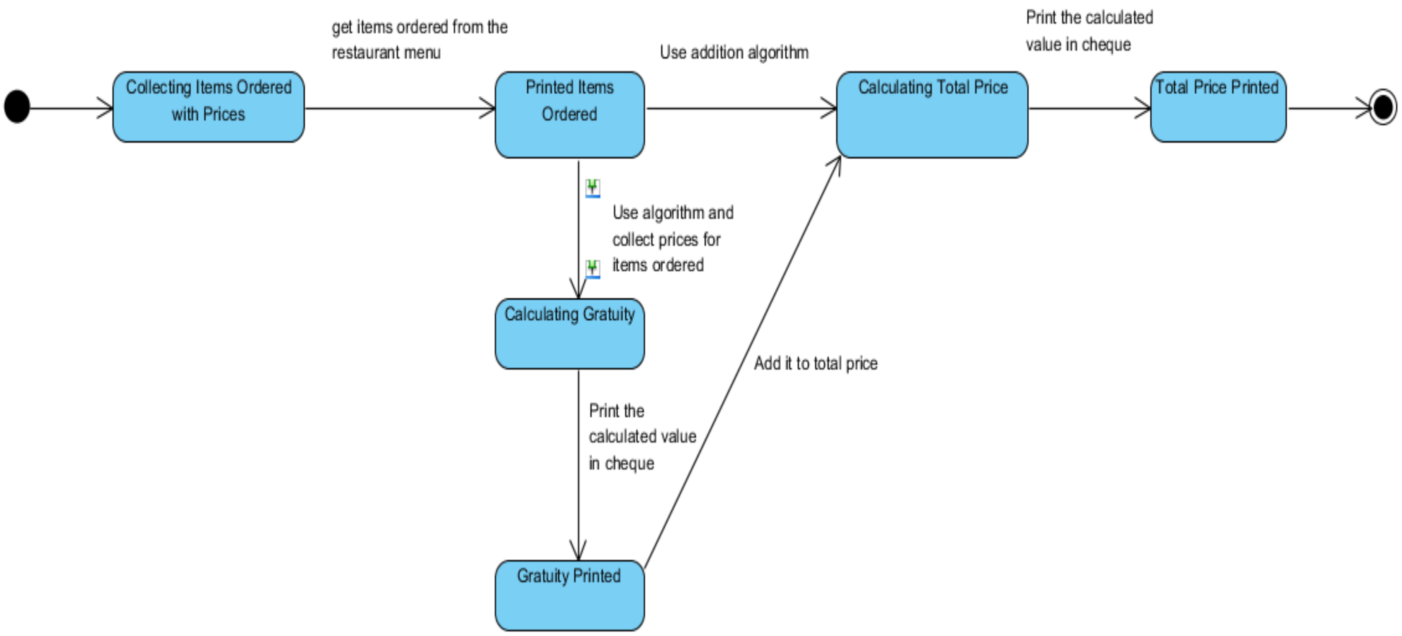
DeleteCategory

Test-case Identifier:	TC-10
Use Case Tested:	UC-5, alternate scenario
Pass/Fail Criteria:	System passes if the chef is successfully able to delete a category in the restaurant menu.
Input Data:	Delete command.
Test Procedure:	**Expected Result:**
Step 1. Login to chef's interface and select the "Create Restaurant Menu" option.	System displays existing menu categories (appetizers, lunch, specials, etc) (if any) and gives the option to "Delete" next to each category. At the beginning of list is the "Add New Category" option. Suppose categories do exist.
Step 2. Select the "Delete" button next to the category wished to be deleted.	System displays pop-up "Are you sure you want to delete category: X?", where X is a category name. System give options "Yes" and "No".
Step 3. Select "Yes".	System deletes the category and returns to category list.

We will also test for the bartender use cases, but there is no need to write test cases for these because the use cases are very similar to UC-5 ChangeMenu. So, the test cases are similar to TC-9 and TC-10.

B) Unit Tests

1. Cheque

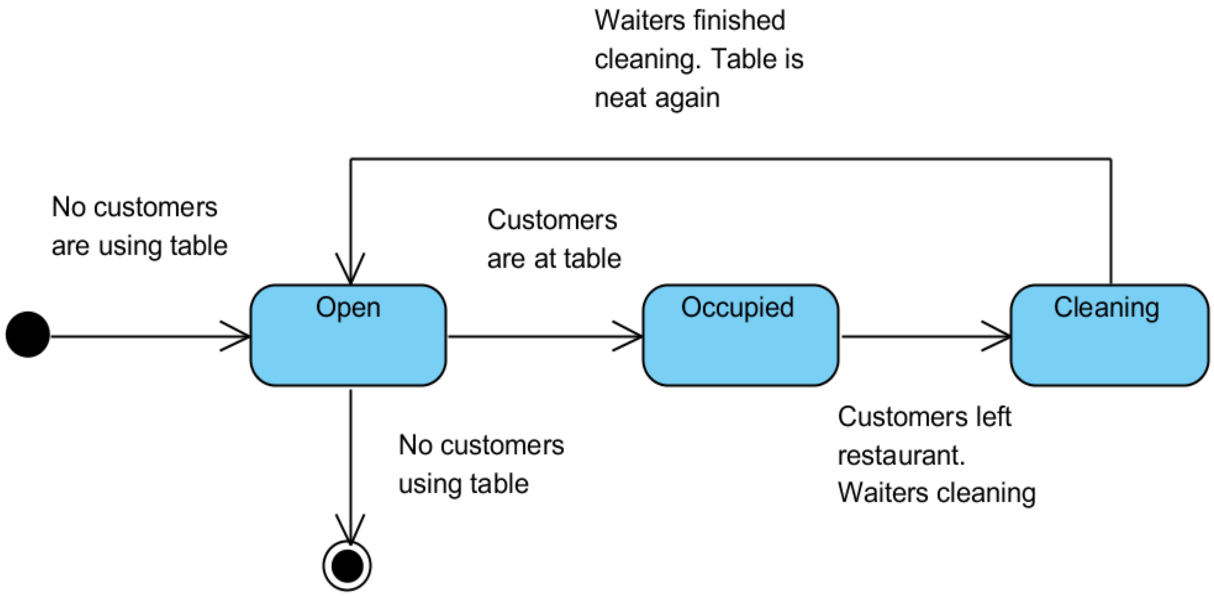

get items ordered from the restaurant menu
Use addition algorithm
Print the calculated value in cheque
Collecting Items Ordered with Prices
Printed Items Ordered
Calculating Total Price
Total Price Printed
Use algorithm and collect prices for items ordered
Calculating Gratuity
Add it to total price
Print the calculated value in cheque
Gratuity Printed

This is the state diagram for the Cheque class. This shows how the cheque for the restaurant customer is calculated and printed on the screen. This includes the total price and gratuity that the customer must pay before leaving. To test all states of this class, we will use the following method calls as described in the class diagram:

getTotal()
setGratuity(total, gratuity, order)
getGratuity()
printCheque()
setPaid(paid)
getPaid()

First, the system collects all ordered items and prices that the customer ordered from the restaurant menu database. It then calculates the total price by adding all prices for each food item and then sets the gratuity rate on the total price. getGratuity() method call simply prints the gratuity rate it calculated. Then it prints the cheque with the ordered items, the total price, and gratuity all in one screen. Finally, setPaid(paid) which returns a boolean value sets the variable paid either 1 or 0. 1 being paid and 0 being not paid. After the customer pays, the variable paid becomes 1 and the "Customer Paid" is printed in the order history.

2. Table

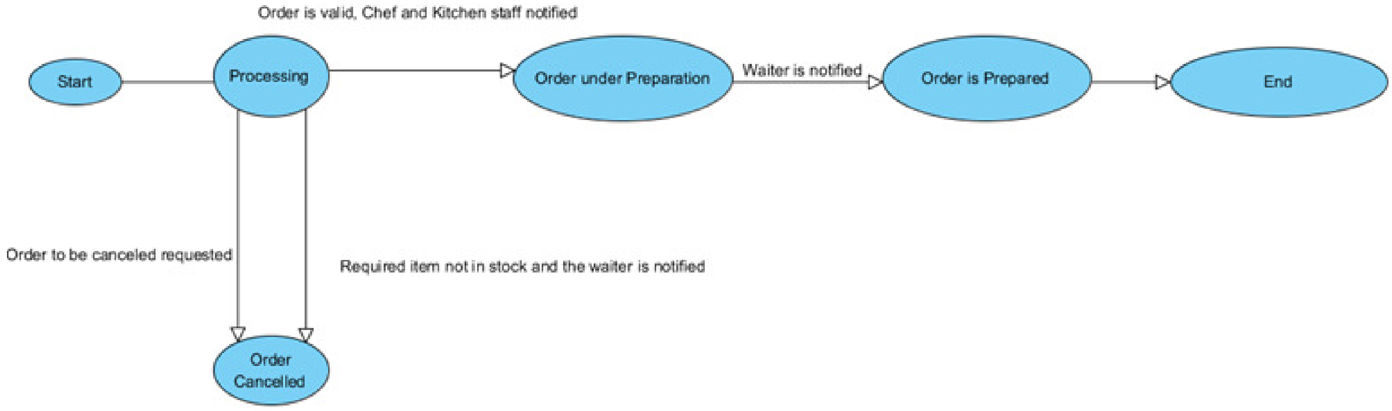

Waiters finished cleaning. Table is neat again
No customers are using table
Customers are at table
Open
Occupied
Cleaning
No customers using table
Customers left restaurant. Waiters cleaning

This is the state diagram for the Table class that shows the status of the restaurant table in the system. The following method calls will be used to test every behavior of the Table class:

getid()
getStatus()
setStatus(status)

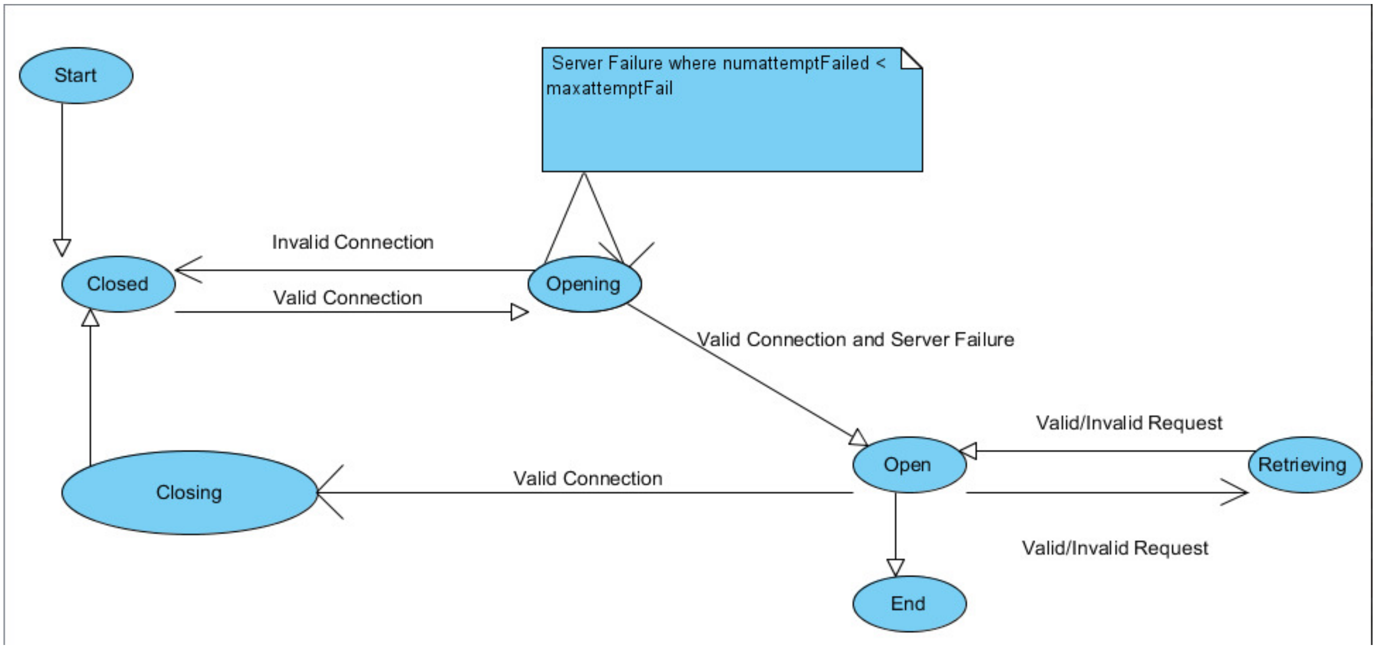
First, the system gets the table number by calling the getid() function. Once the table number is acquired, the status of the table can be 1 of 3 values. We will use enumeration to assign integer values to the status of the table: 1=Open, 2=Occupied, 3=Cleaning. Finally, we will use getStatus to print the status of the table onto the screen.

3. OrderFood


Order is valid, Chef and Kitchen staff notified
Start
Processing
Order under Preparation
Waiter is notified
Order is Prepared
End
Order to be canceled requested
Required item not in stock and the waiter is notified
Order Cancelled

For efficiency, first we test to see if orders have been cancelled. We test the first invalid call with invalidOrder. It is a Boolean test, so if the call is true the order is valid, otherwise it is invalid. Invalid cases can be when the ordered dish does not have an ingredient in stock, or if the order is cancelled. At this stage in the test, the waiter is notified. In addition, because the dish has not yet been prepared, the customer can request the order to be cancelled. The option to cancel an order after this stage will not be given.

4. CreateWebpage



First, the database connection is closed. When a method (ex, openConnection) is called, the state goes to opening. If the connection is invalid, the state goes from opening to closed. There is a limited number of attempts to contact the server. If the connection is valid, it will go to the state open, otherwise, it will go to the state closed.

When a connection is established, we test to see if we can request a query with a method. This method goes from the state open to retrieving, and then reverts to open. It does not matter if the query is valid or invalid because an SQL database will always return a request.

The closing connection is also tested with another method (ex, closeConnection). This method will move the state from open to closing to closed. The state will always go from closing to closed because the database allows safe connection closings and the server closes the connection after any inactivity.

C) Integration Testing

For our system, we can implement both top down integration testing and bottom up integration testing. An instance where top down testing can be used is where the system locks out an unauthorized user. A request to the user for a user id and password is given. When an authorized user submits the correct password to the user id, the system grants the user access. However, if an unauthorized user attempts to gain access by repeatedly submitting an incorrect password, the system will then lock that terminal and record the intrusion attempt into a log. The manager can have access to this log.

Cases where we can use bottom up testing is where we test individual classes that are independent of each other. For many cases, our system has lower level components that are maintained by controllers, so top down testing isn't the best form of integration testing. After each leaf class is tested, we test the next level of the hierarchy and its leaves. An additional advantage to this type of testing is that if an error in testing occurs in a higher level class, bottom up testing helps us to find the error in one of the lower level classes. Because the classes are independent of each other, we can narrow down which of the lower level class contains the problem and search that level's hierarchy instead of a parallel class's hierarchy.

7. Project Management and Plan of Work.

A) Project Coordination

Part 1:

- Rohit Lakshmanatirthakatte opened up a fresh Google Drive document particularly for Report 2 Part 1, and reminded all team members of the due date. Here, we simply built up on the document by adding our assigned use case interaction diagrams.
- Yehuda Cohen arranged a Google+ Hangout for everyone to discuss each team member's responsibilities and who's doing which interaction diagram.
- We also discussed questions and concerns need to be addressed on Facebook chat.
- Finally, Rohit reformatted the entire report as Garamond 11-pt font black text, made a pdf document, and submitted to his Dropbox.

Part 2:

- Rohit Lakshmanatirthakatte opened up a fresh Google Drive document particularly for Report 2 Part 2 and copied and pasted the information from Part 1. Rohit also reminded all team members of the due date. Then, the entire team built up of the information required for part 2 starting from the section "**2. Class Diagram and Interface Specification**".
- Before writing the report, everyone decided to have a Google+ Hangout to divide up the work, who was responsible to which part. The meeting date and time was set up on Facebook chat and by considering everyone's convenience.
- During the Google+ Hangout, Rohit recorded a note of tasks (who was doing what) using the Notepad feature, and posted this on Google Drive for everyone to see.
- Finally, Rohit reformatted the entire report as Garamond 11-pt font black text, made a pdf document, and submitted to his Dropbox.

Part 3:

- Rohit Lakshmanatirthakatte opened up a fresh Google Drive document particularly for Report 3 and Shivani Sethi along with Rohit copied and pasted the information from Part 2. Shivani also updated the Table of Contents section and wrote down the headings for the new sections of Report 3
- Shivani organized a Google+ Hangout and invited people to join. Shivani divided the remaining work among the group members and noted down the responsibilities of each of the group members in a table.
- Finally, Shivani formatted the entire report Garamond 11-pt font black text and sent it to Rohit so that he could submit it via Dropbox.

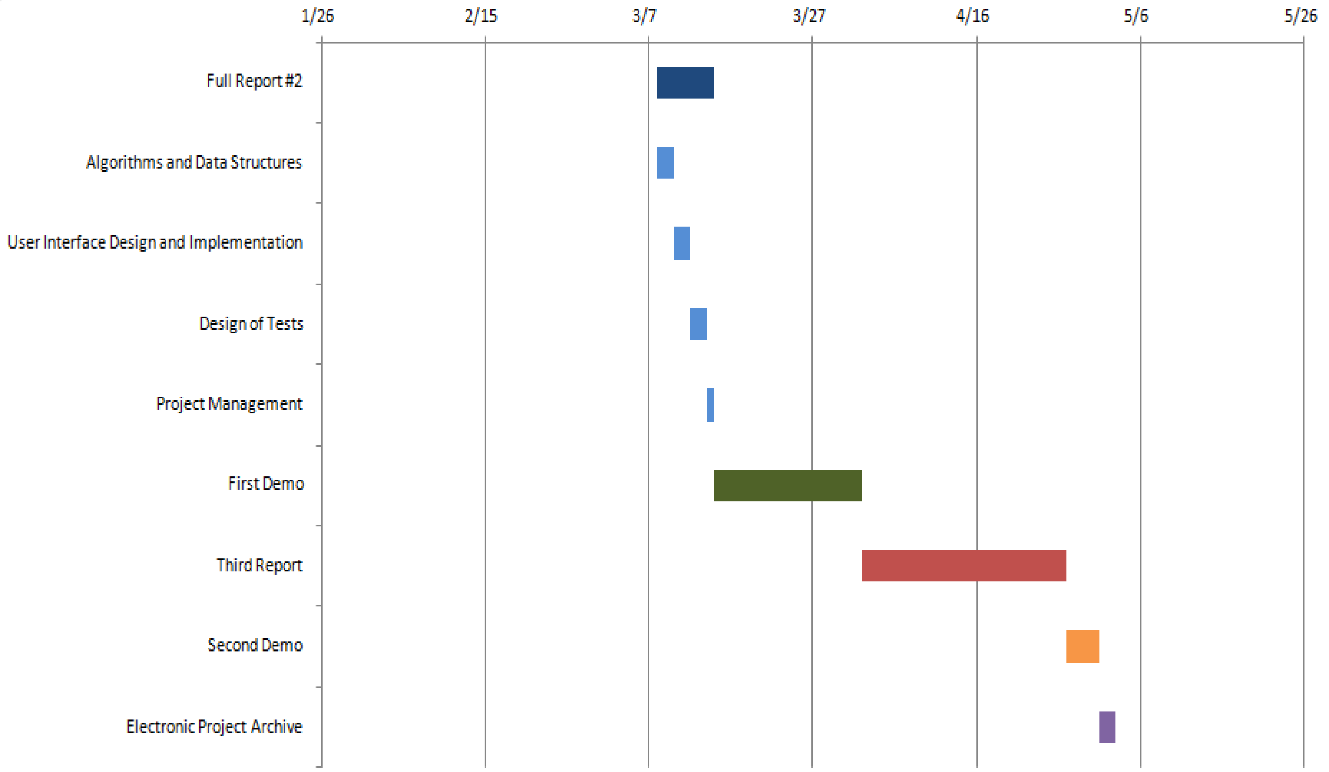
B) Issues Encountered during Project Management

One of the main issues in project management was dividing up the work more equally so that everyone had a chance to work on a significant amount of work and contribute effectively. We dealt with this problem by looking at the breakdown of points in the diagram for report 2 and trying our best to assign an equal number of points to each person. We tried to spread out the work so that everyone got to learn from each other and work together on multiple sections. People were happier that they got a fair chance at contributing so they did more work and were more productive.

C) Progress Report

Of the 71 user stories documented in the first report, we have finally chosen 30 that we will implement for sure in our final design, with an additional 26 that we will implement if time permits. Those 56 user stories have been organized into a backlog by order of priority in the previous report. The various items of this report, such as the class diagrams, database tables, and user interface designs show how we plan to put into operation those user stories that we have selected. Our next step is to derive and employ the actual code from these designs, and thereafter we will go through all of our test cases to ensure our project is ready for the first demo.

D) Plan of Work



E) Breakdown of Responsibilities

The first 30 user stories defined in report one are divided below between the six group members. Each group member is entirely responsible to his/her assigned groups. These items include only the first 30 items from the work backlog: the items identified as essential to the GravyXpress system. Five user stories have been assigned to each of the six group members.

Assignments are as follows:

Yehuda

1	ST-M-1	I want to create a subdomain within the GravyXpress web application specific to my restaurant, so that my restaurant's services are available over the internet.	9
2	ST-G-1	I want to be able to login and logout of GravyXpress securely, so I and only I have access to the areas of GravyXpress that concern me.	3
20	ST-M-2	I want to be able to alter my restaurant's contact information & hours of operation from my dashboard, so that those who visit my restaurant's subdomain always receive up to date information.	2
5	ST-M-10	I want a restaurant's menu that I can modify at a moments notice, so that adding/removing items from my menu as well as changing their price info is not a hassle.	5
12	ST-M-11	I want to enable and disable items from my menu, so that I can perform temporary alterations to my menu.	2

Abdul

24	ST-V-2	I want to learn about GravyXpress and the service it provides.	1
25	ST-V-1	I want to view an attractive web page that looks professional and draws me in.	7
26	ST-Ch-4	I want to modify existing items on the menu, changing their name or altering their details if necessary, so that I can always change or improve each item on the menu.	4
27	ST-Ch-2	I want to add and delete new items to and from the menu, so that I can offer variety to my customers.	5
29	ST-Ch-5	I want to disable and enable items on the menu, so that they can be temporarily available or unavailable to customers (as needed).	3

Amizan

4	ST-M-5	I want to create new user accounts for my employees, so that they may perform their relevant duties through GravyXpress.	5
21	ST-M-6	I want to alter an employee's permissions, so that they only retain access to the services on GravyXpress that concern them.	2
3	ST-M-4	I want to add and remove tables to my restaurant, specifying the number of seats they have, so that customers are only offered tables with enough seats for their party.	3
11	ST-W-1	Once I am logged in, I want to see only information pertaining to the customers I am assigned to, so that I don't serve other waiter's customers inadvertantly.	1
16	ST-W-2	In my own profile, I want to be able to view which tables (by table number) I am assigned to, so I know exactly which customers to serve.	2

Shivani

6	ST-C-1	I want to order food quickly and efficiently through a web page, so that I can order without the help of a waiter.	8
9	ST-C-5a	I want to be able to view the cheque, so that I can be aware of the amount of money I am spending.	5
14	ST-C-4	I want to be able to signal a waiter while seated at a table, so that I don't need to wave my hands about flailing for attention.	1
15	ST-C-12	I want to be able to cancel or change selected orders if they haven't been sent to the kitchen, so I can continue changing my mind until the kitchen has begun preparing.	3
30	ST-W-6	I want to be able to modify a table's orders, so the restaurant customers can tell me to order for them if they want.	5

Rohit

28	ST-W-11	I want to seat a customer at an available table, so that the system can keep an updated account of the status of the tables in the restaurant.	2
7	ST-K-1	I want to fetch the orders from the customers, which are stored in the order queue, and add them to the kitchen queue, so that I am always aware of all of the items I am currently cooking.	5
8	ST-K-3	I want to mark an order item as ready and see it removed automatically from the kitchen queue, so that my list of tasks remains uncluttered.	5
13	ST-K-6	I want to see how many orders await in the order queue waiting to be fetched, so that I know how busy the restaurant is and pace myself accordingly.	2
17	ST-K-5	I want the system to send a signal to the waiter once an order has been marked ready, so that customers receive their food in a timely fashion.	4

Nabil

10	ST-W-4	I want to be able to see what items my assigned table(s) are ordering, so I can bring the correct orders to the table.	4
18	ST-W-5	I want to see any customer-help signals, so I can attend to them without delay.	1
19	ST-W-7	I want to be able to simply view my assigned table's check, so I know what they ordered and how much they owe.	3
22	ST-W-9	After cleaning the tables, I want to be able to change the status of my tables to ready so the system knows the customer has left.	4
23	ST-W-10	After a customer has left, I want to see that my table responsibility has been deleted in my profile, so I don't get confused about which table to serve next.	4

8. References

1 The *Software Enginnering* textbook by Ivan Marsic. Link at: http://www.ece.rutgers.edu/~marsic/books/SE/book-SE_marsic.pdf
2 Microsoft Visual Paradigm for UML 10.1 Community Edition downloadable at http://www.visual-paradigm.com/download/vpuml.jsp
3 Group 2's project from Spring 2012 as a reference to see how interaction diagrams are created: http://www.ece.rutgers.edu/~marsic/books/SE/projects/Restaurant/2012-g2-report3.pdf
4 Group 11's project from Spring 2012 as a reference for Parts 2 and 3: http://www.ece.rutgers.edu/~marsic/books/SE/projects/Restaurant/2012-g11-report3.pdf
5 A Youtube video about how to draw sequence diagrams: http://www.youtube.com/watch?v=18_kVlQMavE
6 A tutorial on using Visual Paradigm for database design: http://knowhow.visual-paradigm.com/database-design/design-database-with-schema/
7 Microsoft's Website on SQL servers: http://msdn.microsoft.com/en-us/library/ms143506.aspx
8 Microsoft's Website on screen and resolution settings: http://windows.microsoft.com/en-us/windows-vista/getting-the-best-display-on-your-monitor