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

Working demo: http://gravyxpress.herokuapp.com/

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

Revision History:

Version No.	**Date of Revision**
1	05/05/2013
2	05/12/2013

Individual Contributions Breakdown:

Abdul Rattu:

- Contributed in **Summary of Changes** explained what has been changed from our original plans.
- Contributed in updating the section of **Customer Statement of Requirements** regarding chef requirements.
- Updated **Glossary of Terms** with few more terms we used in the project reports.
- Revisited on-screen appearance requirement, analyzed previously drawn handmade user stories sketches under **System Requirements**.
- Contributed in traceability matrix - mapping use cases with user stories under **Functional Requirements Specification**.
- Contributed to explain use case five (UC-5) under **Functional Requirements Specification.**
- Contributed in system operational contracts under **Domain Analysis**, created contract for "about restaurant"
- Contributed in explaining Design Pattern under **Interaction Diagrams**.
- Added **OCL Contract Specification** for my user stories of chef under the section of **Class Diagram & Interface Specification**
- Explained mapping subsystems to hardware under **System Architecture & System Design**.
- Revisited all **Graphical User Interface Requirement** to make sure if there is anything needs to be updated in future.
- Contributed in designing and explaining **User Interface Design and Implementation.** The evolutions of our design.
- Contributed in writing and explaining test cases such as the chef menu update test case under the section **Design of Test.**
- Contributed in the section of **History of Work, Current Status and Future Work**. Explained what needs to be done in future for integration.
- Contributed in **Reference** section, provided all the references I used to make my parts of the reports and project.
- Worked accordingly on teams leader's requests in **Project Management.**

Yehuda Cohen:

- Relevant parts from Report 1 and 2 as explained in the respective contribution breakdowns. This included the vast majority of user stories, the class diagram, and all of the *system* sequence diagrams.
- Redid user story sequence diagram and explanation for Restaurant Creation to eliminate page maker class by using Play! Framework's template engine. Updated user stories and descriptions accordingly.
- Modified Menu Builder description to eliminate the capability to edit a pre-existing menu item to make dashboard interface cleaner. Updated user stories and descriptions accordingly.
- Effort Estimation for Restaurant Creation.
- Described MVC design pattern and it's nativity to Play! framework.
- Updated UI Design specification based on what was implemented.
- Identified areas of improvement in the future for dashboard (asynchronous requests) and HTML subset suggestion to prevent malicious HTML injection.

Nabil Ali:

- Contributed in **Summary of Changes** explained changes

- Helped Rohit edit the ManageOrder interaction diagram
- Edited hardware requirements tables and descriptions
- Reviewed Algorithm and Data Structures sections and added a few additional details
- Edited Use Case tables for Waiter and Restaurant Customer by highlighting cases with red, yellow, and green statuses
- Edited the size points for the Use Case Waiter table, Bartender table, and Restaurant worker
- Revisited personal sequence diagrams from previous diagrams and verified them
- Verified Matlab code, added more comments and adjusted acii spaces for a better formatted output
- Contributed in the **History of Work , Current Status and Future Work** sections. Described plans for the future
- Contributed to effort estimation table and traceability matrix for Waiter
- Added additional resources used under References (Gantt Chart videos)

Shivani Sethi:

- Updated all the requirements for the customer user stories
- Updated the summary of changes section
- Edited/Revised the Glossary of Terms
- Updated the Stakeholders/Actors and Goals section
- Added descriptions to the System Requirements section
- Updated the Operations/Contracts section for the Customer
- Updated the Customer User Stories
- Updated the Create Webpage Interaction Diagram
- Updated the System Architecture Section
- Updated and revised the Traceability Matrix to include the Customer User Stories
- Updated the Algorithms and Data Structures Sections
- Worked on Project Management according to Team Leaders requests
- Updated the Code for the Customer User Stories so that it could be implemented on the Play Framework
- Made the Customer and Call Waiter Interfaces for the corresponding User Stories

Amizan Jaleel:

- Did nearly all of the **operation contracts** under **Domain Analysis**, as it was not complete when report 1 was submitted.
- Updated the work backlog for the user stories that I was assigned to implement; namely: ST-M-5, ST-M-6, ST-M-4, ST-W-1, ST-W-2
- Contributed to the traceability matrix under **Functional Requirements Specification** that maps use cases with user stories.
- Contributed to the details of the **design patterns** used under **Interaction Diagrams**, as I argued the benefits of the MVC framework and how they benefit the project.
- Contributed to the section **History of work, Current Status, and Future Work**, where I detail what has been completed of my user stories as well as future work.
- **Effort estimation** for serve tables.
- Updated the **summary of changes**
- Added the **OCL** under **Class Diagram and Interface Specification** that pertains to my implementations

Rohit Lakshmanatirthakatte:

- Wrote all of the **Table of Contents**.
- Contributed to the **Summary of Report Changes** by adding 10 bullet points to the list most of them being about the Kitchen Staffer.
- **Customer Statement of Requirements:** Wrote the 2nd bullet point of **Waiting Staff** section. Rewrote 3rd, 4th, and 5th bullet points of the **Kitchen Staff** section.
- **System Requirements/User Stories:** In the **As a Waiter...** section re-wrote ST-W-7, ST-W-9, and ST-W-10 user stories. In the **As a Kitchen Staffer Worker...** section re-wrote ST-K-1, ST-K-3, ST-K-5, and ST-K-6 user stories. For the user stories mentioned, I also wrote new size points depending on one difficult they were to implement.
- **Work Backlog:** Updated the backlog by writing block numbers 7, 8, 13, 17, 22, 36, and 37.
- **Functional Requirements Specification:** For the **Fully Dressed Use Cases** section, I re-wrote all of the ManageOrder use case (UC-4) according to how I implemented it. In the **System Sequence Diagrams** section I updated all of the **Prepare Order as a Kitchen Staff Worker** diagram and provided a description of the diagram. In the **Traceability Matrix Mapping Use Cases with User Stories** I added and mapped ST-K-1, ST-K-3, ST-K-5, and ST-K-6.
- **Effort Estimation using Use Case Points:** I did all of the **ManageOrder (Kitchen Staff Worker) UC-4** section until I calculated the duration of the use case.
- **Domain Analysis:** In **Domain Model**, re-wrote bullet point number 4. I also did the entire **Concept Definitions, Association Definitions, Attribute Definitions, and Traceability Matrix.**
- **Interaction Diagrams:** I re-did the **ManageOrder Use Case** interaction diagram and provided a brief description of it.
- **Class Diagram and Interface Specification:** I updated the Class Diagram with the proper attributes and operations of the KitchenQueue, OrderQueue, and Tables classes. I re-wrote the Data Types and Operation Signatures of the just those 3 classes mentioned above. I redid the traceability matrix that maps domain concepts with the software classes.
- **User Interface Design and Implementation:** I added my interfaces for my Kitchen, Order Queue, Waiter Notifications, and Tables Creation/Status page.
- **Project Management:** I managed the project by continuously reminding other team members about the due date of this report on Facebook group chat, formatting this report as 11-point Garamond, compiling this report, and submitting to my Dropbox. I also organized a Google+ hangout to discuss the requirements for Report 3.

Table of Contents:

Summary of Report Changes:

- Redesigned restaurant creation process, eliminating the need for a page-maker class -- All pages are served by dynamically rendered HTML templates. The Play! Framework makes this easy with its scala based templating engine.
- Eliminated the editing of menu items feature due to dashboard clutter. Upon implementation, we felt that the need to edit a menu item was not necessary given the easy addition and removal of menu items.
- Designed chef user stories in different database asp.net, it was supposedly to be implemented in Play! framework.
- Used adobe dreamweaver for the website which is different than our original plans of using CSS.
- Operation contracts were added under the domain analysis, as they were absent from our first submission of report 1
- Use cases of the work backlog have been updated
- All System Requirements related to the Kitchen Staff Worker has been updated according to how it was implemented.
- All System Requirements related to the Customer have been updated according to how it was implemented
- The Fully Dressed Use Case for MangeOrder (UC-4 Kitchen Staff Worker) has been updated.
- The Sequence and Interaction Diagram for the ManageOrder (UC-4) has been updated.
- A Traceability Matrix mapping User Stories with Use Cases has been added.
- A new section Effort Estimation using Use Case Points has been added.
- In the Domain Analysis section, Concept, Association, Attribute Definitions and Traceability Matrix have been updated and added. The matrix shows how use cases map to domain concepts.
- A new class diagram has been posted where the KitchenQueue, OrderQueue, and Tables classes were updated.
- Data types and operation contracts were updated for the KitchenQueue, OrderQueue, and Tables classes.
- A new subsection called the Object Constraint Language was added to this document.
- Some new interfaces of the actual implementation of the software was added.
- A new section, History of Work, Current Status, and Future Work has been added.

1. Customer Statement of Requirements:

As an experienced manager of a restaurant, I feel that effective communication is perhaps the quintessential ingredient to any successful operation involving collaboration. The restaurant that I manage, as is the case in many other privately-owned restaurants, is comprised of a great number of interacting teams who must always be in perfect sync with one another to ensure productivity levels are optimal. Slow communication and miscommunication are two barriers that my restaurant cannot put up with in this communication age.

What I want is a software system, that effectively manages communication between the different teams involved in my restaurant. These teams are:

- Restaurant Customers
- Managerial Staff
- Waiting Staff
- Kitchen Staff (the cooks)
- Chefs
- Bartenders

I want this restaurant management software to be as accessible as possible, such that any user using any electronic device with a web browser will be able to access this software, and enable all of the aforementioned teams to interact with each other using this software. As such, I would like this product to be developed as a webapp. The system should enable me to create profiles for each restaurant staff position, so I can easily manage the different teams about their activities throughout the day.

I have taken the time to note down the inefficiencies that I would like to eliminate from my restaurant with an automated system. These identified inefficiencies are:

1 My customers, who want to preorder food, often have to deal with busy phone lines with possibly long waiting times.
2 My customers are usually in a hurry and would like to order before arriving at the restaurant.
3 Sometimes my customers order takeouts but do not arrive to collect them, thereby wasting the restaurant's valuable resources.
4 My customers often do not have the freedom to select their own table.
5 Tracking down a waiter to alter or cancel an order that has not yet been sent to the kitchen often is a hassle for my customers.
6 My customers are often unable to gain the attention of a waiter.
7 Calculating gratuity manually is an unnecessary inconvenience for my customers.
8 Waiting for the cheque can be inconvenient for my customers.
9 On busy days, my customers must wait a long time before a table becomes available and they are seated.
10 For me, keeping track of which tables are reserved and the durations of their reservation requires unnecessary manual monitoring.
11 There are frequent miscommunications regarding orders between my customers, the waiters, and kitchen staff.

12 While some of my waiters are fairly idle, others are often either overwhelmed with work.
13 Waiters waste time and paper rifling through a notepad to access information regarding a table's orders.
14 Waiters must frequently return to the kitchen to determine the status of my customer's orders.
15 We usually have to maintain a manual queue of precedence to ensure customers who order food first are served food first, which result in mixups.
16 If ingredients run short or menu items are altered by me or the chef, the rigid paper menus that we have cannot be updated quickly and conveniently.
17 Separating orders intended for the bar from those intended for the kitchen is inconvenient for my waiters as it wastes time.
18 I have trouble keeping up with all activities occurring in the restaurant, sometimes leading to delays of my customer's orders.

I would like these inefficiencies to be addressed with the development of this management software and eliminate most paper-pen transactions. As a user of this software, my goal is to have an effective communication system within the restaurant so mishaps, as described above, will not not occur and can be avoided in the future.

The software system should allow six different users (as mentioned above) to successfully communicate with each other allowing the following abilities for each user:

Customer:

- I want my customers to be able to access my restaurant web app with any electronic device that has access to the internet.
- My customers can enter the food items they wish to order along with the quantity of those food items. The items ordered are then displayed on the results page in a list format. They are then manually entered into the Order Queue.
- Customers should only be able to make changes to their order when the order is in the phase before it has entered the kitchen queue.
- I want some way to allow those customers, who ordered online, to be able to convey that information when they enter the restaurant, such that they will immediately be directed to their chosen table.
- On the online webpage, customers should have the options to either reserve a table or order take-out, and a time limit of when they should be at the restaurant before their reservation is cancelled, in order to save restaurant resources.
- If customers simply walk into the restaurant without online reservations, those customers should have the ability to choose their own available table right at the front desk, enter the number of people in the group into the system, and be assigned a single waiter for that table until the customers leave the restaurant.
- Once at the table, my customers should still be able to use all features that are online if they want to order/cancel more food using their own mobile device or a tablet pc set at their table.
- Instead of calling their waiter repeatedly to check on their orders, my customers should get to know the status of their food using the software.

- When the time comes to pay the bill, the customers should be able to use a gratuity calculator provided by the software.
- In case the customer wants to use the drive-thru service, I want them to be able to do so using the online service or using the tablet pc mounted at the drive-thru window.

Managerial Staff:

- I want the ability to create a website specifically for my own restaurant, so I can add features available for my customers and staff.
- I want the ability to create profiles for each of the six restaurant positions complete with proper security (username/password).
- I want full control over the customer online payment feature, such as enabling and disabling it.
- I want features to add a floor plan of my restaurant (complete with tables and chairs), access and handle employee pay stubs, access customer order history and their cheques, view customer feedback, and manage the restaurant menu online.
- I also want to track the status of ingredients in stock and also have the ability to update them manually.
- I want to view the popularity of menu items.
- I want a central announcement board so I can convey information to my employees easily.
- I want to add promotional ads to my website as well as have the ability to send promotions to customers who subscribe for them.
- I want to run statistical reports of the restaurant activities and profits at the end of each working day.

Waiting Staff:

- When a customer "checks in" at the restaurant, I want the software system to automatically assign a waiter to the preferred table.
- When the waiter seats a customer at a chosen table, I want to allow that waiter to update the status of the table in the system such as "Open", "Occupied", "Needs Cleaning".
- I do not want any waiter to be idle or have too much workload, so I want the system to assign waiters to tables depending on the number of customers.
- All waiters will be given special smart phones so they can access their assigned table's orders and order status.
- That being said, each waiter should be given a special access number so that they can only access the tables they are assigned to.
- Each of my waiters should be able to see a customer-calling signal if a customer signals to them.
- An assigned waiter, when signaled, should be allowed to alter the restaurant customers' order, such as adding, deleting, or changing an item in the order.
- Each waiter should be able to access the central dashboard for announcements made by the manager.

Kitchen Staff:

- I want a separate profile for my kitchen staff, complete with security features such as a special access number.
- I will be having only one touch-screen tablet pc handled by one person in the staff, so I want all orders to appear on that screen.

- I want all orders coming from the customers to be collected into one big list, so that there is some organization.
- I want my kitchen staff workers to have the freedom to select orders that they want to prepare first. There may or may not be a first-come-first-prepared order.
- I also want the software system to allow initial orders (orders not yet chosen by the kitchen staff) to be updateable by the customer before it becomes a permanent order (order chosen by kitchen staff for preparation).
- The kitchen staff should also be able to read any customer preference-notes.
- After orders have been completed, the system should allow the kitchen staff worker to delete that order and then signal the assigned waiter to collect the order.
- Each kitchen staff worker should be able to access the central dashboard for announcements made by the manager.

Chefs:

- The chefs should also have a separate profile with a special access number.
- I want the chefs to have the ability to change/add items on the menu, including the names of items, the prices, and ingredients count.
- They should also be able to remove an item from the menu due to ingredient shortage. This way, the customer will not be able to accidentally order it.
- Chef should also be able to update the current menu with the description of the ingredients as well as the prices of the items.
- Each chef should be able to access the central dashboard for announcements made by the manager.

Bartenders:

- My bartenders also should have a separate profile with a special access number.
- Bartenders, just like the chefs, should have the ability to change/add drinks on the menu, including names of drinks, the prices, and ingredients count.
- They should also be able to remove a drink from the menu due to ingredient shortage. This way, the customer will not be able to accidentally order it.
- The software system should separate beverage orders from the food orders as the customer hits the order button.
- Each bartender should be able to access the central dashboard for announcements made by the manager.

2. Glossary of Terms:

1) **Bartenders** - Control the drinks menu and may alter it, along with notifying waiters of drink orders.

2) **Centralized Communication System (CCS)** - A communication system where all the users connect to a central server which stores all the information for the system

3) **Chef** - Manages and directs the kitchen staff and may also alter the restaurant menu.

4) **Clock In/Clock Out** - Each employee must enter the time that they start working (clock in) and the time that they stop working (clock out). This will help to run the payrolls since many employees get paid by the hour.

5) **Restaurant Customer** - Orders food and services from the restaurant and pays for these services either online or in the restaurant. Can eat at restaurant, get take-out, or pick up at a drive-thru.

6) **Dynamic** - Flexible and changeable. Dynamic items are designed to be altered quickly and with ease.

7) **Floor Plan Layout** - Shows all the tables and chairs in the restaurant along with their corresponding table status.

8) **Inefficiency** - Does not produce the desired effect and is not an economical solution. Can make processes slow or inconvenient

9) **Interface** - Visual on computer, tablet, or phone that allows for user interaction with the GravyXpress system. For instance, customers can place orders and pay bills via the interface.

10) **Kitchen Staff** - Can fetch items from the order queue and put them in the kitchen queue. Can also mark items that have been cooked as complete and remove them from the kitchen queue.

11) **Manager** - The manager controls what features the restaurant will offer, such as the online payment/ordering module and bar module. The manager can also control the number of tables in the restaurant, view employee information, view popularity of items, view customer feedback and records of past orders. The manager also has permissions of all players in the restaurant.

12) **Managerial Trend Digest** - Available from the managerial dashboard. The trend digest gives the manager an overview of the statistics of his restaurant. This digest includes statistics such as popular dishes and average customers per day. (The development of this digest depends largely on whether enough time is available to do so.)

13) **Menu** - List of items available in the restaurant. Sides, such as french fries, can also be selected along with the main dishes. The menu is visible online at GravyXpress, and customers can directly place their order online.

14) **Module** - A part of a program that carries out a specific function. It may be used alone or with other modules in the program.

15) **Payment** - Upon completion of the meal or upon receiving take-out, customers can request the cheque, add gratuity using a gratuity calculator, and pay online through PayPal. Payment can also be handled traditionally with credit card or cash by calling a waiter.

16) **Payroll** - The salary of each employ. Different employees are paid different amounts. For instance, chefs make more money than waiters do.

17) **Queue (in terms of kitchen and order queues)** - For instance, when orders are made by restaurant customer, they are sent to an order queue. When the kitchen staff worker is ready to prepare the order, the worker drags the order from the order queue into kitchen queue. The order that is easier to make is prepared first. It is NOT the case that the first order placed is prepared first.

18) **Reservation System** - Customers may reserve a table via GravyXpress. Once they arrive at the restaurant, they will be directed to an automatically assigned table. They will also have the option of selecting another open table from a floor plan schematic.

19) **Semi-customizable** - Can be partly customized by the users to fit their specific needs. For instance, the manager can choose the number of tables that are available in the restaurant, and thus help customize the user interface for the floor layout.

20) **Subdomain** - A domain that is part of a larger domain.

21) **Table Status** - A table can either be free (green color), dirty (yellow color), or occupied (red color). The status of each table along with the waiter assigned to the table is made visible on the floor plan layout of GravyXpress.

22) **User Role** - A user role describes the relationship between users and the system.

23) **Visitor** - A general term referring to any person visiting any page on GravyXpress. Could be a person who browses the internet and stumbles upon GravyXpress or a person looking for career opportunities in a restaurant.

24) **Waiter** - Must attend to their assigned tables. They have access to each of the tables' orders, can see the status of their orders, and can modify orders if restaurant customers prefer the waiter to order for them.

25) **GravyXpress** - A general restaurant cloud where restaurant owners can go and create their own restaurant website according to their needs.

3. System Requirements/User Stories:

As a Restaurant Customer...

Identifier	User Story	Size (points)
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
ST-C-2	I want to make a reservation at a restaurant quickly and efficiently over the internet so that I know a table will be waiting for me when I arrive.	5
ST-C-3	I want to select my table from a floorplan of available (changes from green to red) tables, so I am seated at the table that suits me best.	4
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
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
ST-C-5b	I want to pay my cheque online securely, so that I can pay remotely or from my own mobile phone without even taking out my wallet.	4
ST-C-6	I want to be able to pay my gratuity without having to manually calculate percentages.	2
ST-C-7	I want to post feedback and provide ratings regarding customer service, food quality and overall experience.	1
ST-C-8	I want to subscribe to updates informing me of specials and other relevant notifications about the restaurant.	2
ST-C-9	I want to have the option of take-out when I order, so I don't need to wait for a table on busy days.	2
ST-C-10	I want to order at a drive-thru, so I don't need to enter the restaurant on busy days.	5
ST-C-11	I want to see an estimated waiting time to select an available seat on busy days (when no seats are available), so that I know when to order take-out or order at a drive-thru.	5
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
ST-C-13	I want to be able to add side notes to selected orders, so that I can tell the kitchen staff about my ingredient preferences.	2
ST-C-14	I want to receive an order number when I order online, so that I don't need to go through the trouble of registering a user account.	1
ST-C-15	I want to be given a time limit of my reservation, so I can plan my trip to the restaurant accordingly.	1

As a Restaurant Manager...

Identifier	User Story	Size (points)
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
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
ST-M-3	I want to upload an image of a floor plan to my restaurant from my dashboard so that my customers can later select their table directly from the floor plan.	4
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
ST-M-5	I want to create new user accounts for my employees, so that they may perform their relevant duties through GravyXpress.	5
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
ST-M-7	I want to be able to keep my food inventory up to date, so that I am always in touch with my stock of produce.	4
ST-M-8	I want to be alerted when an ingredient in my inventory is running short, so that I know I need to buy more.	5
ST-M-9	I want to post new job openings, so that potential employees know when jobs are available at my restaurant.	1
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
ST-M-11	I want to enable and disable items from my menu, so that I can perform temporary alterations to my menu.	2
ST-M-12	I want to manage my business financial account information including employee pay stubs and corporate food payments, so I can prepare paychecks.	6
ST-M-13	I want to create and post employee schedule so that my employees know when they are expected to work.	4
ST-M-14	I want to post important information on the public message-board so that employees are well informed.	3
ST-M-15	I want to send promotions to customers via email/text, so customers can take any opportunities of discount.	2

ST-M-16	I want to view the records of my previous business, such as customers' order history so that I can utilize my restaurant's history to improve its future.	4
ST-M-17	I want the application to keep track of the day of the week, so that I can assign specific specials on given days.	3

As a Waiter...

Identifier	User Story	Size (points)
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
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
ST-W-3	In the general waiter interface, I want to see any announcements made by the manager, so I am well informed about any news/activities.	3
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
ST-W-5	I want to see any customer-help signals, so I can attend to them without delay.	1
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.	3
ST-W-7	I want to be able to acknowledge all orders that I have delivered, so that I can clear those responsibilities from my interface.	1
ST-W-8	I want to be able to simply view my assigned table's check, so I know what they ordered and how much they owe.	4
ST-W-9	After the customers have paid, I want the system to alert me that the customers, at my table, have left (table status gets changed from occupied to dirty), so I know when to clean the table for future customers.	4
ST-W-10	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
ST-W-11	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.	3

As a Kitchen Staff Worker...

Identifier	User Story	Size (points)
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	7

	currently cooking.	
ST-K-2	I want to see which orders are take-out and drive-thru orders, so that I know to package them in disposable containers.	4
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.	3
ST-K-4	I want to see notes attached by customers to their orders, so that I know to prepare their food in a particular manner.	5
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.	3
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.	1

As a Chef...

Identifier	User Story	Size (points)
ST-Ch-1	I want to be able to modify the supply of ingredients for each menu item, so that a manager knows when to replenish the stock.	4
ST-Ch-2	I want to add new items to the menu, so that I can offer variety to my customers.	5
ST-Ch-3	I want to delete items from the menu, so that unsuccessful meals become unavailable to customers.	3
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
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

As a Bartender...

Identifier	User Story	Size (points)
ST-B-1	I want to add new drinks to the menu, so that I can offer variety to my customers.	5
ST-B-2	I want to delete drinks from the menu, so that unsuccessful drinks become unavailable to customers.	3
ST-B-3	I want to modify existing drinks on the menu, changing their name or altering their descriptions if necessary, so that I can always change or improve each item on the menu.	5
ST-B-4	I want to disable and enable drink items on the menu, so that they can be temporarily	2

	available or unavailable to customers (as needed).	
ST-B-5	I want to fetch the next drinks to be mixed from the order queue, so that I can make drinks at my own pace.	4
ST-B-6	I want to mark a drink as ready, and remove it from my own queue, so that my list of tasks remains uncluttered.	3
ST-B-8	I want system to send alerts to waiters to fetch ready drinks, so that customers receive their drinks in a timely fashion.	4
ST-B-9	I want to be able to modify the supply of ingredients, so that a manager knows when to replenish the stock.	4

As a General Restaurant Worker...

Identifier	User Story	Size (points)
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.	4
ST-G-2	I want to view my working schedule, so that I know when I need to be available for work and when I have vacation hours.	2
ST-G-3	I want to put a schedule swap request with other employees based on their availability in emergency/non-emergency situations.	3
ST-G-4	I want to upload my tax documents such as W2 & W4 forms so that the manager can view and maintain them for his records.	3
ST-G-5	I want to view my pay stubs as well as enter bank account information, so the manager can directly deposit my paycheck to my bank account. The system will not handle direct deposits. It will only provide information to manager where to deposit.	2
ST-G-6	I want to submit a day off request to my manager, so that I don't have to search for the manager to do so. Also, I can have a counter of the amount of sick and off days I have remaining.	2

As a Visitor...

Identifier	User Story	Size (points)
ST-V-1	I want to view an attractive web page that looks professional and draws me in.	7
ST-V-2	I want to learn about GravyXpress and the service it provides.	1
ST-V-3	I want to be able to see postings of job opportunities at the restaurant, so I can contact the manager to apply for the job.	2

Work Backlog:

#	Identifier	User Story	Size (points)
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.	4
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, and so that the restaurant's maximum capacity for customers is known.	3
4	ST-M-5	I want to create new user accounts for my employees, so that they may perform their relevant duties through GravyXpress. I also want this ability to create user accounts to be restricted only to me, so that no user can create an employee on the website without my permission.	5
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 is not a hassle.	7
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
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.	7
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.	3
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. Note that the cheque covers the cost of all the people who ordered on the table.	5
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
11	ST-W-1	I want to be able to see the number of customers at a table that I am assigned to.	1
12	ST-M-11	I want to effortlessly enable and disable items from my menu, so that I can perform temporary alterations to my menu.	2
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.	1
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. The waiter should be able to see the tables that need attention on a dashboard so that they can attend to their needs.	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
16	ST-W-2	In my own profile, I want to be able to view which tables (by table number) I am assigned to, and I want to be assigned a table only when I am not assigned to another.	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.	3
18	ST-W-5	I want to see any customer-help signals, so I can attend to them without delay.	1
19	ST-W-8	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
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
21	ST-M-6	I want to be able to assign my employees specific roles, and that each role will have responsibilities and permissions associated with it.	2
22	ST-W-10	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.	5
23	ST-W-11	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
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 new items to the menu, so that I can offer variety to my customers.	5
28	ST-Ch-3	I want to delete items from the menu, so that unsuccessful meals become unavailable to customers.	2
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
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
31	ST-M-13	I want to create and post employee schedule so that my employees know when they are expected to work.	4
32	ST-G-2	I want to view my working schedule, so that I know when I need to be available for work and when I have vacation hours.	2

33	ST-C-2	I want to make a reservation at a restaurant quickly and efficiently over the internet so that I know a table will be waiting for me when I arrive.	5
34	ST-C-5b	I want to pay my cheque online securely, so that I can pay remotely or from my own mobile phone without even taking out my wallet.	5
35	ST-C-7	I want to post feedback and provide ratings regarding customer service, food quality and overall experience.	1
36	ST-W-9	After the customers have paid, I want the system to alert me that the customers, at my table, have left (table status gets changed from occupied to dirty), so I know when to clean the table for future customers.	7
37	ST-W-7	I want to be able to acknowledge all orders that I have delivered, so that I can clear those responsibilities from my interface	1
37	ST-M-14	I want to post important information on the public message-board so that employees are well informed.	3
38	ST-W-3	In the general waiter interface, I want to see any announcements made by the manager, so I am well informed about any news/activities.	3
39	ST-C-9	I want to have the option of take-out when I order, so I don't need to wait for a table on busy days.	3
40	ST-C-15	I want to be given a time limit of my reservation, so I can plan my trip to the restaurant accordingly.	1
41	ST-C-13	I want to be able to add side notes to selected orders, so that I can tell the kitchen staff about my ingredient preferences.	2
42	ST-K-4	I want to see notes attached by customers to their orders, so that I know to prepare their food in a particular manner.	2
43	ST-M-16	I want to view the records of my previous business, such as customers' order history so that I can utilize my restaurant's history to improve its future.	4
44	ST-M-9	I want to post new job openings, so that potential employees know when jobs are available at my restaurant.	1
45	ST-V-3	I want to be able to see postings of job opportunities at the restaurant, so I can contact the manager to apply for the job.	2
46	ST-K-2	I want to see which orders are take-out and drive-thru orders, so that I know to package them in disposable containers.	3
47	ST-C-6	I want to be able to pay my gratuity without having to manually calculate percentages.	2
48	ST-C-11	I want to see an estimated waiting time to select an available seat on busy days (when no seats are available), so that I know when to order take-out or order at a drive-thru.	5

49	ST-C-14	I want to receive an order number when I order online, so that I don't need to go through the trouble of registering a user account.	1

On-Screen Appearance Requirements:

Below are the hand drawn sketches for our user stories that are directly related to our user interface. We drew the sketches only for the user stories where we need to give preliminary interface ideas to our customer. Most of the user stories are directly or indirectly related with each other in sense of user interface, hence sketches for those stories are not individual but has been drawn together and being represented in one window.

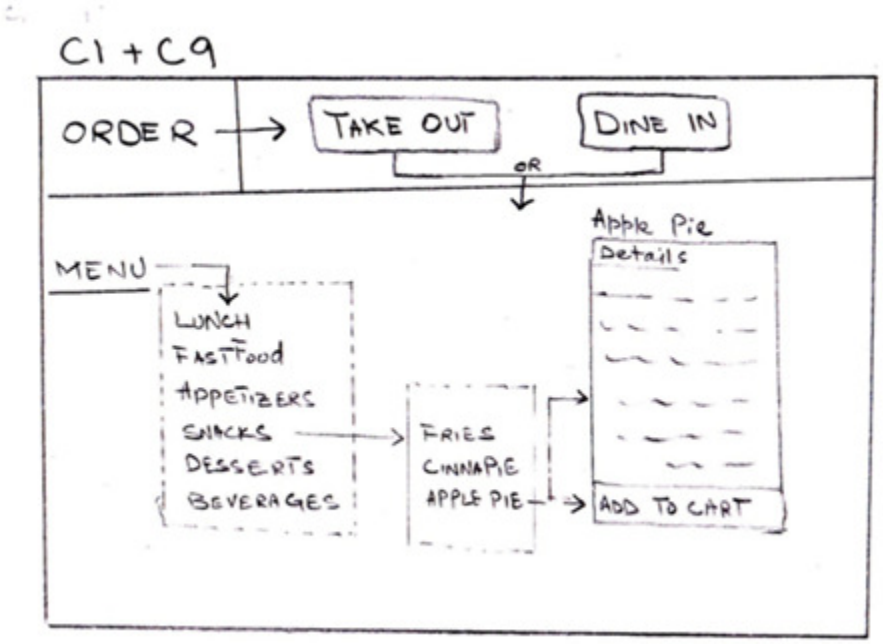
In general, the emphasis in our User Interface design relates to enabling customers to interact with GravyXpress using touch devices such as smart phones and tablets. We have shied away from interfaces that require text entry as much as is possible because simple button clicking is easier on touch devices.

We have also tried to balance the amount of information available from each webpage, so that each page is mobile friendly while at the same time users do not need to navigate to other pages frequently. With this methodology in mind, we have produced the sketches below and elaborated each in some detail.

Story ST-C-1 & ST-C-9

Sketch below illustrate our customer user stories 1 & 9 where customer can easily navigate through the menu and place the order without the help of waiter as well as S/he will have choice of "take out" and "dine in".

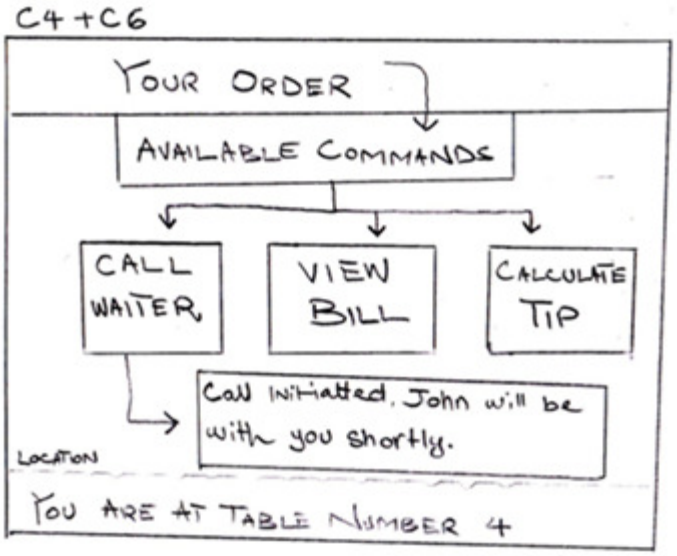
In order to facilitate ease of use, the menu will be divided into three sections. The first section will enable a customer to select a menu section. The second section will enable a customer to select a menu item within that section. The third will contain the details for that particular menu item.



Story ST-C-4 & ST-C-6

Sketch below illustrate our customer user stories 4 & 6 where customer will have option to call waiter and ease of calculating the gratuity.

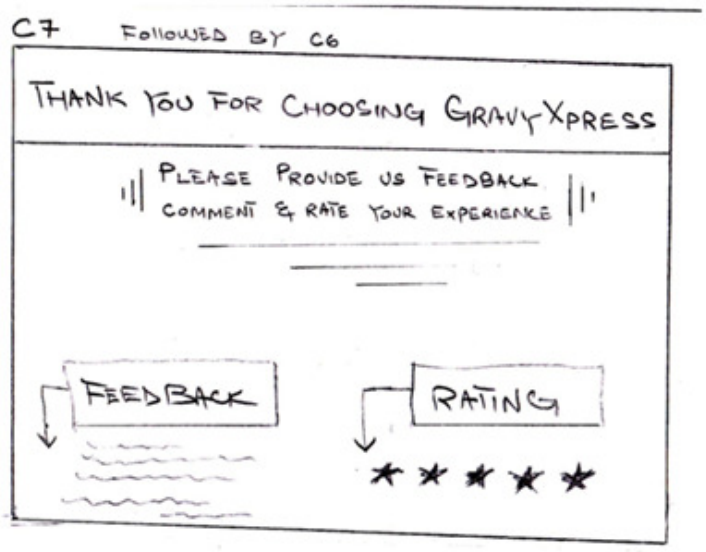
These stories represent the basic interface a customer at a GravyXpress restaurant will be presented with. Upon selecting "Your Order", the customer will be presented with the option to call a waiter, view the cheque or calculate a tip. This User interface will be simple with large buttons that are easy to see and easy to press.



Story ST-C-7

Sketch below illustrate our customer user story 7 where customer can easily provide feedback and ratings to the restaurant.

The "GravyXpress" in the header of this page will be replaced with whatever the restaurant's name is. This page represents our ideal farewell page. The customer will be prompted to enter feedback, but this will occur at the bottom of the page in a non-intrusive fashion.

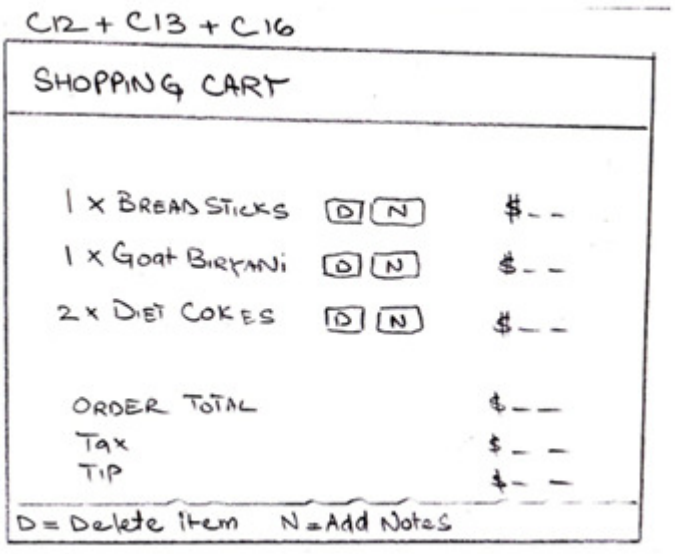


Story ST-C-12, ST-C-13 & ST-C-16

Sketch below illustrate our customer user stories 12, 13 & 16 where customer will have option to view his/her shopping cart, remove item and add notes to the individual items.

This would be the equivalent of a checkout screen for our online ordering system. It gives the customer a chance to review his/her order before it is placed, as well as modify it as necessary. It also includes allows them to add notes to each individual order, which would then be available to the kitchen staff, as was specified in the user stories.

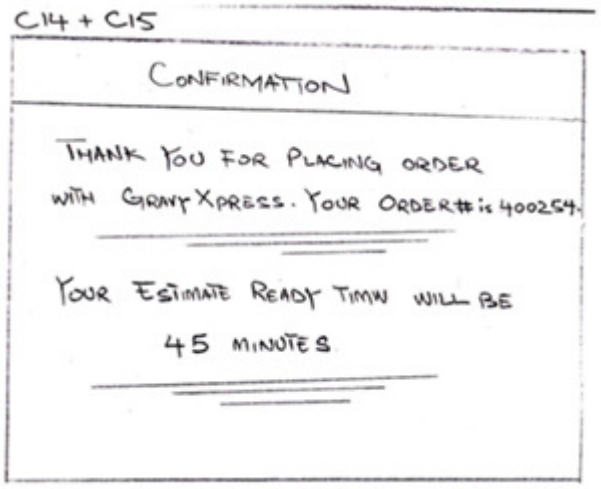
Also important when viewing this is the idea that the shopping cart will be tied to a cheque, constantly keeping the customer aware of what he is spending.



Story ST-C-14 & ST-C-15

Sketch below illustrate our customer user stories 14 & 15 where customer will get the confirmation number after placing the order as well as estimate ready time.

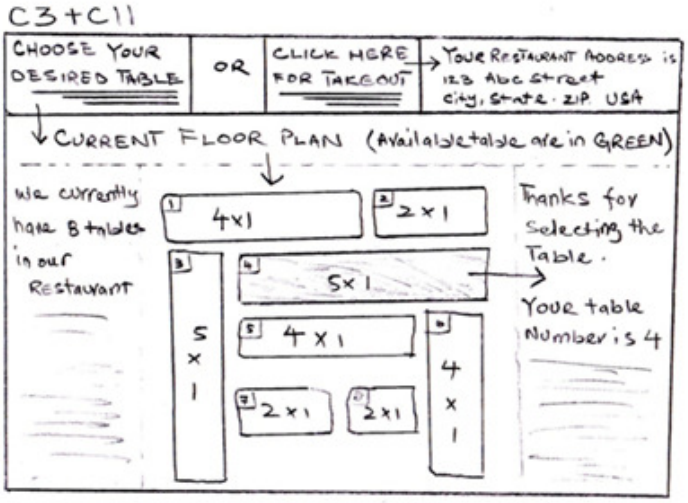
After checkout, when the customer finalizes the order, this would be the last screen they would see in the transaction. It is important for each customer to both confirm their order and plan their time of arrival according to the estimation.



Story ST-C-3 & ST-C-11
Sketch below illustrate our customer user stories 3 & 11 where customer will have flexibility to choose his/her choice of table as well as S/he can also change his/her mind to "take out".

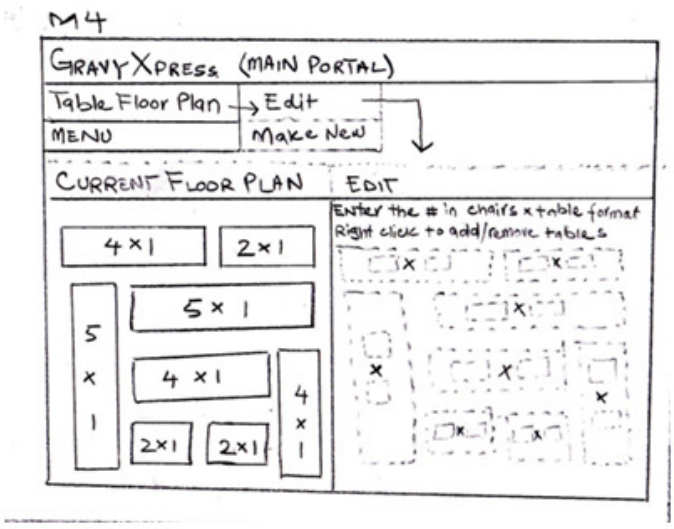
Please note that while we like the idea of creating a floor map immensely, it will in all likelihood be one of the last features we implement. This is because of two reasons. The first of these is that we don't believe it is at the core of the communication system that GravyXpress is designed to implement. The second is that creating a cloud interface to enable a manager to map his own floor plan to the system's list of tables is a project in and of itself.

If we do get around to implementing it, we want to provide a clean interface that enables a customer to select a seat with ease, with the tap of a touch screen.



Story ST-M-4

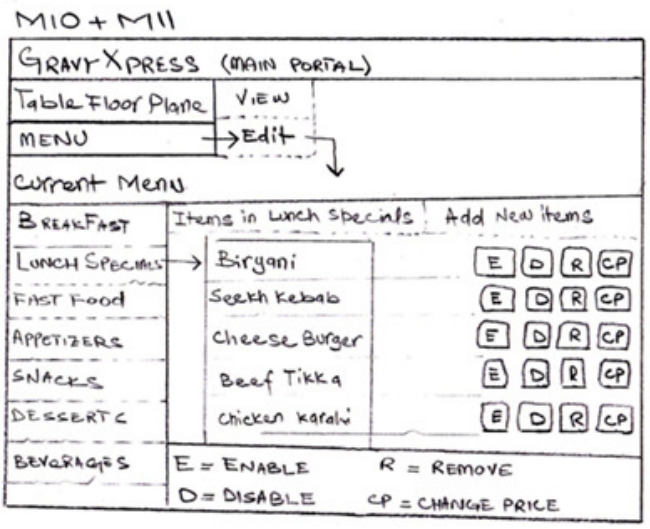
Sketch below illustrate our manager user story 4 where manager can edit/update the floor plan efficiently. In the below diagram, when "Make New" is selected a manager will be presented with a dialogue to upload an image. Clicking different locations on the image will map those portions of the image to the tables in the system. This simple system is modeled around facebook's image tagging system. This is preferable to a jQuery drag and drop HTML format that dynamically alters the Document Object Model because it enables restaurant owners to upload floorplans which are abnormal such as restaurants with multiple floors with ease.



Story ST-M-10 & ST-M-11

Sketch below illustrate our manager user stories 10 & 11 where manager can edit the menu items, enable/disable them, removing of items as well as changing the price of the items.

It is necessary to give the manager full control over the selection of items on the menu. Each item is categorized by its type, and the manager can alter the properties and price for each one. The manager can also add or remove items, enable and disable them.



4. Functional Requirements Specification:

Please be aware that the user stories tables as defined above consist of every user story we would like to make a part of GravyXpress. For now, we will elaborate only on those user stories highlighted in green to begin with. This order of precedence is apparent in the work backlog where we have arranged User Stories as we believe they should be implemented. We believe the greeen stories represent the core of what GravyXpress adds to the table. The uniqueness of GravyXpress is not to automate restaurant management, or to provide a hub for customers to review restaurants. Such projects have been created before, and well at that.

Rather, the innovation of GravyXpress lies in its ability to provide restaurant managers all over the world an easy cloud web service to serve as a real time distributed communication system in their restaurants. It is the instant notifications between the Kitchen staff, waiters, and customers that together make GravyXpress the great service that it is.

In this light, we will not elaborate the user stories related to anything other than the focal core of GravyXpress. In the true style of Agile Development, such documentation can be produced as we are able to integrate these user stories into an already functional cloud based restaurant communication system.

A. Stakeholders:

One of the main categories of stakeholders in this system will be end users, such as restaurant managers, waiters, chefs, kitchen staff, bartenders and visitors. They are mainly interested in the routine system functions. The restaurant customer is another type of stakeholder. Customers will interact with the system just as much as the end users, but for a different purpose - to utilize the facilities provided by the restaurant. Last but not least, the developers, software architects, system analysts and project manager are stakeholders who will design and implement the system.

B. Actors and Goals:

1. Restaurant Customer:

a) Role - Interacts with GravyXpress to order food directly, online, at a drive-thru, or by take-out.

b) Type - Initiating actor.

c) Goal - To order food and services quickly and efficiently.

2. Managerial Staff:

a) Role - Uses GravyXpress to manage all activities of restaurant as described in the user story.

b) Type - Initiating actor.

c) Goal - To customize the system for their specific restaurant, overlook the operations of the restaurant, keep track of their employees' information (e.g. pay), and change features of the restaurant such as the floor layout and menus.

3. Waiting Staff:

a) Role - Serves the restaurant customers and is notified of their orders through GravyXpress.

b) Type - Participating (supporting) actor.

c) Goal - To maintain servicing of different tables for customers by bringing food to customers, cleaning tables and updating table statuses. Can also view checks and order totals.

4. Kitchen Staff:

a) Role - Cooks the customer's food by **choosing orders from the Order Queue and notifying the assigned waiter when finished.**
b) Type - Participating (supporting) actor.
c) Goal - To quickly and efficiently prepare the food ordered by customers and view the Order and Kitchen Queues.

5. Chefs:
a) Role - Manages the kitchen staff, modifies the restaurant menu, and maintains ingredients.
b) Type - Initiating actor.
c) Goal - To modify and/or delete and add items from the menu. To notify manager of inventory.

6. Bartenders:
a) Role - Serves drinks to customers by viewing drink orders.
b) Type - Participating (supporting) actor.
c) Goal - To control the bar menu and inventory for the bar. To make drinks from the order queue and notify the waiter when the drinks are ready to be served.

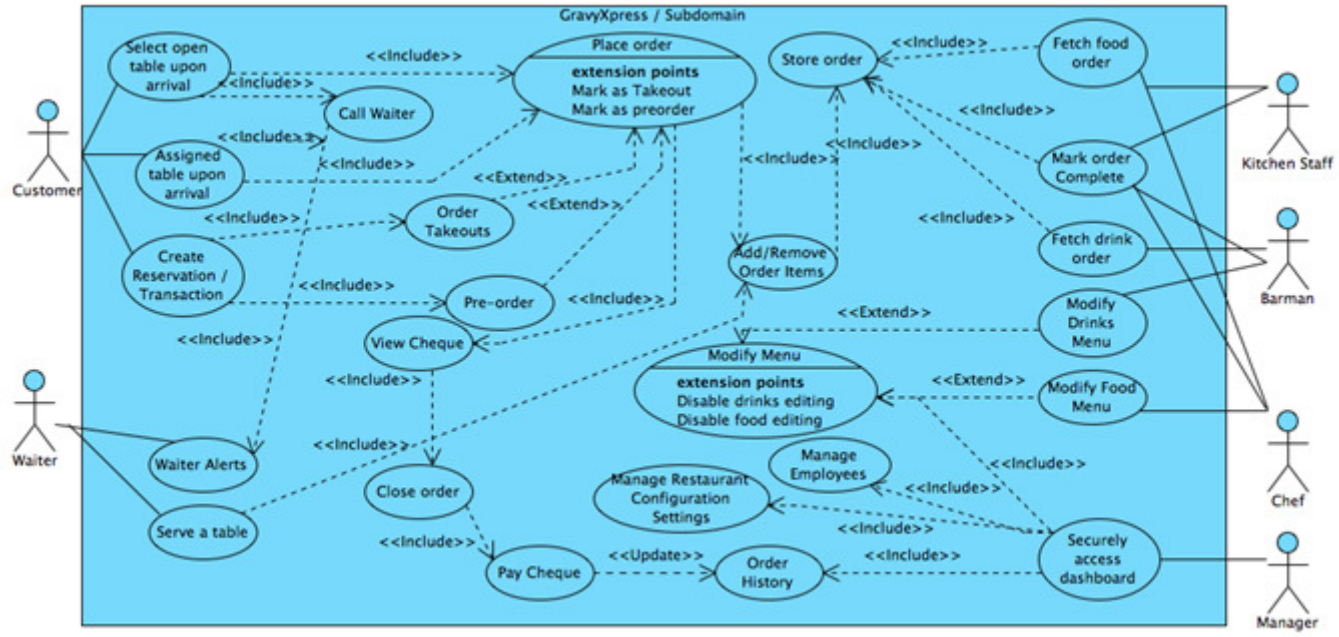
7. General Restaurant Worker:
a) Role - Keeps the restaurant working and maintains his or her own accounts on the system.
b) Type - Initiating or participating depending on what position worker holds in the restaurant.
c) Goal - To complete the required training, keep track of work schedule/holidays and acquire pay stubs/tax documents.

8. Visitors:
a) Role - Accidentally stumbles upon GravyXpress while browsing the web. Or he or she is a potential applicant searching for career opportunities at a restaurant.
b) Type - Participating (offstage) actor.
c) Goal - To see how attractive the webpage is and how GravyXpress works. Or to find a job at a restaurant.

C. Use Cases:

i) Casual Description - Since the development team produced user stories instead of stating system requirements, the user stories will serve as casual description of the use cases.

ii) User Stories Diagrams -



iii) Fully-Dressed Description –

Customer:

Use Case UC-1:	**OrderFood (walk-in customer)**
Related User Stories:	ST-C-1, ST-C-4, ST-C-12 to ST-C-13, ST-C-16 to ST-C-17
Initiating Actor:	Restaurant Customer
Actor's Goal:	To view the menu, select preferred items, and order the items
Participating Actors:	Screen display, tablet or mobile device, user interface, waiter (optional)
Preconditions:	Screen displays main GUI with options to view menu and order.
Postconditions:	Come back to main menu or show status of food.

Flow of Events for Main Success Scenario:

→ 1. Restaurant customer sees the table number on Main Menu and selects "View Menu" option.

← 2. System displays menu categories (drinks, appetizers, specials, lunch, dinner, etc.)

→ 3. Restaurant customer selects a category.

← 4. System displays all items in that category, price of the item, attach note option, and the "Add to Cart" option.

→ 5. Restaurant customer selects the "Add to Cart" option to the item.
← 6. System counts the number of items and calculates the total cost, both at the corner of the screen.
← 7. System automatically sends the orders from the cart to the Kitchen's Order Queue.
← 8. 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".

← 9. When system displays "Food is Ready and Arriving", system goes back to main menu after 1 minutes.

Flow of Events for Extensions Alternate Scenarios:

→ 1a. Restaurant customer selects the "Call Waiter" option for help in using GravyXpress.
← System displays pop-up message "Calling Waiter" until waiter arrives to help customer. Message closes after 1 minute and system goes back to window previously visited.

→ 3a. Restaurant customer selects the "Return to Main Menu" option to go back.

← System goes back to Main Menu.

→ 5a. Restaurant customer selects the "Attach Notes" option to write quick notes about the item.

← System displays a small textbox next to the item.

→ Restaurant customer selects the submit button to attach the note to the item.

← System displays "Note Attached" and returns to the menu where the restaurant customer left off.

→ 7a. Restaurant customer selects the "View Order" option to review list of items ordered.

← System shows all items selected by customer including the "Cancel" option for each item.

→ 7b. Restaurant customer selects "Cancel" option to remove selected items.

← System removes selected items.

→ 8a. Restaurant customer selects the "Remove Items" option when status reads "Order Queue".

← System displays the list of items ordered by restaurant customer with options to cancel selected items.

→ Restaurant customer selects the "Add More Items" option in the status window.

← System goes to the menu categories page

→ Customer repeats the process of adding items to cart.

← System sends new orders to the Order Queue in the kitchen.

→ 8b. Restaurant customer selects the "Remove Items" option when status reads "Kitchen Queue".

← System displays error message "Order is already being prepared. Cannot cancel order!" and goes back to order status page.

Manager:

Use Case UC-2:	**CreateWebpage**
Related User Stories:	ST-M -1, ST-M-2, ST-M-5, ST-M-13
Initiating Actor:	Managerial Staff
Actor's Goals:	To create a subdomain within the GravyXpress web application specific to my restaurant, post my restaurant name and hours of operation, and set up accounts for my employees.
Participating Actors:	Screen Display, tablet or mobile device, user interface, dashboard, Manager
Preconditions:	Screen displays main GUI with options to add a subdomain within GravyXpress, alter restaurant basic information, and create/maintain employee profiles.
Postconditions:	Come back to main dashboard GUI so that Manager can alter and run restaurant.

Flow of Events for Main Success Scenario:

→ 1. Manager goes on GravyXpress web application and selects "Create my restaurant".

← 2. System asks for name of restaurant, manager name, hours of operation, and address.

→ 3. Manager provides system with restaurant name, manager name, hours of operation and address.

→ 4. 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.

→ 5. Manager asks the system to create a new user account for an employee by selecting "Add employee" on dashboard interface.

← 6. System asks Manager to specify type of employee, pay roll for employee, and work schedule for employee.

→ 7. Manager enters the information for his specific employee into the system.

← 8. System goes back to the dashboard interface.

Flow of Events for Extensions Alternate Scenarios:

Have same flow of events as above up to number 8, but continue with the following steps (Note: the main difference is that multiple employees instead of just one employee can be added).

← 9. Manager asks the system to create a new user account for an employee by selecting "Add employee" on dashboard interface.
← 10. System asks Manager to specify type of employee, pay roll for employee, and work schedule for employee
→ 11. Manager enters the information for his specific employee into the system.
← 12. System goes back to the dashboard interface.

Waiter:

Use Case UC-3:	**ServeTable**
Related User Stories:	ST-W-1 to ST-W-2, ST-W-4 to ST-W-10
Initiating Actor:	Waiter
Actor's Goal:	To maintain servicing of different tables for customers by bringing food to customers.
Participating Actors:	Customers, Kitchen Queue
Preconditions:	Waiter's interface shows status of food for his/her assigned tables.
Postconditions:	The food status becomes "Ready" to deliver to restaurant customer.

Flow of Events for Main Success Scenario:

→ 1. Waiter logs into his/her account and selects the "Assigned Tables" option.
← 2. System displays the table numbers that the waiter 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.

→ 3. Waiter selects one of the table numbers he/she is assigned.

← 4. System displays the order details of the table, including table number, the price of each item, and order status.

← 5. Kitchen queue reports "Order Ready" for a given table number. The entire table row is highlighted in green.

→ 6. Waiter selects the "Acknowledged" button after serving the food to table.

← 7. System changes the status of table to "Served" and removes the highlighting.
← 8. System alerts waiter that table wants to pay by cash. Displays "Payment by Cash" for that table and highlights the row yellow.

→ 9. Waiter selects the "Acknowledged" button after collecting cash and giving receipt to table.

← 10. System alerts waiter that table needs to be cleaned. Displays "Cleaning Required".

→ 11. Waiter selects the "Acknowledged" button after cleaning the table.

← 12. System deletes the table from the list of tables to serve.

Flow of Events for Extensions Alternate Scenarios:
← At any time, the system alerts waiter to assist table by highlighting row red and displaying "Assistance Required" message.

→ Waiter selects the "Acknowledged" button after assisting the table.

← System clears the "Assistance Required" message and removes highlighting.
←2a. The system changes or deletes table order details when order is in "In Order Queue" status and when table wants to add/delete their order.

Kitchen Staff Worker:

Use Case UC-4:	**ManageOrder**
Related User Stories:	ST-K-1, ST-K-3, ST-K-5, ST-K-6, ST-W-7
Initiating Actor:	Kitchen Staff Worker
Actor's Goal:	To fetch orders and add them to kitchen queue and remove completed orders from kitchen queue

Participating Actors:	Customer, Waiter, Chef (offstage)
Preconditions:	Interface shows the Order Queue (with orders from customers) and the Kitchen Queue (may or may not have orders)
Postconditions:	Order has been removed from both Queues and waiter has been notified for pick up.

Flow of Events for Main Success Scenario:

→ 1. Kitchen staff worker approaches screen to view and fetch orders.
← 2. System displays both Order Queue and Kitchen Queue, which are automatically refreshed to account for new incoming orders. Suppose the Kitchen Queue is initially empty.
→ 3. Kitchen staff worker presses the "Transfer to Kitchen Queue" button for each order he wants to prepare from the Order Queue.
← 4. System transfers just the name of the food item to the Kitchen Queue and removes the "Transfer to Kitchen Queue" button for each order pressed to prevent multiple clicks on the same order.
→ 5. Kitchen staff worker presses the "Complete" button after he/she finishes an order.
← 6. System signals the waiter to pick up the order and clears the order from the Kitchen Queue and Order Queue.

Flow of Events for Extensions Alternate Scenarios:

→ 1a. Customer cancels an order.
← b. System refreshes and the order is removed from the Order Queue.
→ 3a. Customer attempts to cancel an order after the order has been moved to the Kitchen Queue.
← b. System does not cancel the order because the "Delete" button in the Order Queue is not there.

Chefs:

Use Case UC-5:	**ChangeMenu**
Related User Stories:	ST-Ch-1 to ST-Ch-4, ST-M-7, ST-M-10
Initiating Actors:	Chefs, Managerial Staff
Actor's Goal:	To add or remove items from the restaurant menu.
Participating Actors:	None.
Preconditions:	User has the "Create Restaurant Menu" screen open.
Postconditions:	Restaurant menu is updated and user can see new changes to the menu.

Flow of Events for Main Success Scenario:

→ 1. Chef or manager selects the "Create Restaurant Menu" option once logged in.
← 2. 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.

→ 3. Categories do exist. Chef or manager selects category to add a new item.
← 4. 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".

→ 5. Chef or manager selects the "Add New Item" option.
← 6. System displays pop-up window with some textboxes to give "Name:", "Price:", and "Inventory Count:" of the new item.

→ 7. Chef or manager enters information and hits the "Add Item" when finished.

← 8. System adds the item into the category list in alphabetical order.

→ 9. Chef or manager presses "Main Menu" when finished.

← 10. System returns to main user interface.

Flow of Events for Extensions Alternate Scenarios:

→ 2a. Chef or manager selects the "Delete" option to delete a category.

← 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".

→ Chef or manager selects "Yes".

← System deletes the category and returns to category list.

→ 4a. Chef or manager selects "Delete Item" next to the item he/she wants to delete.
← System displays pop-up "Are you sure you want to delete item: X", where X is an item name. System give options "Yes" and "No".

→ Chef or manager selects "Yes".

← System deletes the item and returns to items list.

→ 4b. Chef or manager selects "Change Item" next to the item he/she wants to change.
← System displays pop-up window with some textboxes to change "Name:", "Price:", and "Inventory Count:" of the existing item.

→ Chef or manager enters information and hits the "Update Item" when finished.

← System updates the item and returns to the items list.

Bartenders:

Use Case UC-6:	**ChangeDrinks**
Related User Stories:	ST-B-1 to ST-B-3, ST-B-9, ST-M-7, ST-M-10
Initiating Actor:	Bartender, Managerial Staff
Actor's Goal:	To add or remove items from the "Drinks" category in the restaurant menu.
Participating Actors:	None.
Preconditions:	User has the "Drinks" category open.
Postconditions:	"Drinks" category is updated and user can see those changes.

Flow of Events for Main Success Scenario:

→ 1. Bartender or manager selects the "Drinks" category once logged in.
← 2. System displays subcategories of drinks (sodas, juices, caffeine, etc.) (if any) with options to "Change Name" and "Delete" a subcategory next to each. At the beginning of list is the "Add New Subcategory" option.

→ 3. Subcategories do exist. Bartender or manager selects subcategory to add a new item.
← 4. System displays names of all items in that subcategory 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".

← 5. Bartender or manager selects the "Add New Item" option.
→ 6. System displays pop-up window with some textboxes to give "Name:", "Price:", and "Inventory Count:" of the new item.

→ 7. Bartender or manager enters information and hits the "Add Item" when finished.

← 8. System adds the item into the subcategory list in alphabetical order.

→ 9. Bartender or manager presses "Main Menu" when finished.

← 10. System returns to main user interface.

Flow of Events for Extensions Alternate Scenarios:

→ 2a. Bartender or manager selects the "Delete" option to delete a subcategory.
← System displays pop-up "Are you sure you want to delete subcategory: X", where X is a subcategory name. System give options "Yes" and "No".

→ Bartender or manager selects "Yes".

← System deletes the subcategory and returns to subcategory list.

→ 4a. Bartender or manager selects "Delete Item" next to the item he/she wants to delete.
← System displays pop-up "Are you sure you want to delete item: X", where X is an item name. System give options "Yes" and "No".

→ Bartender or manager selects "Yes".

← System deletes the item and returns to items list.

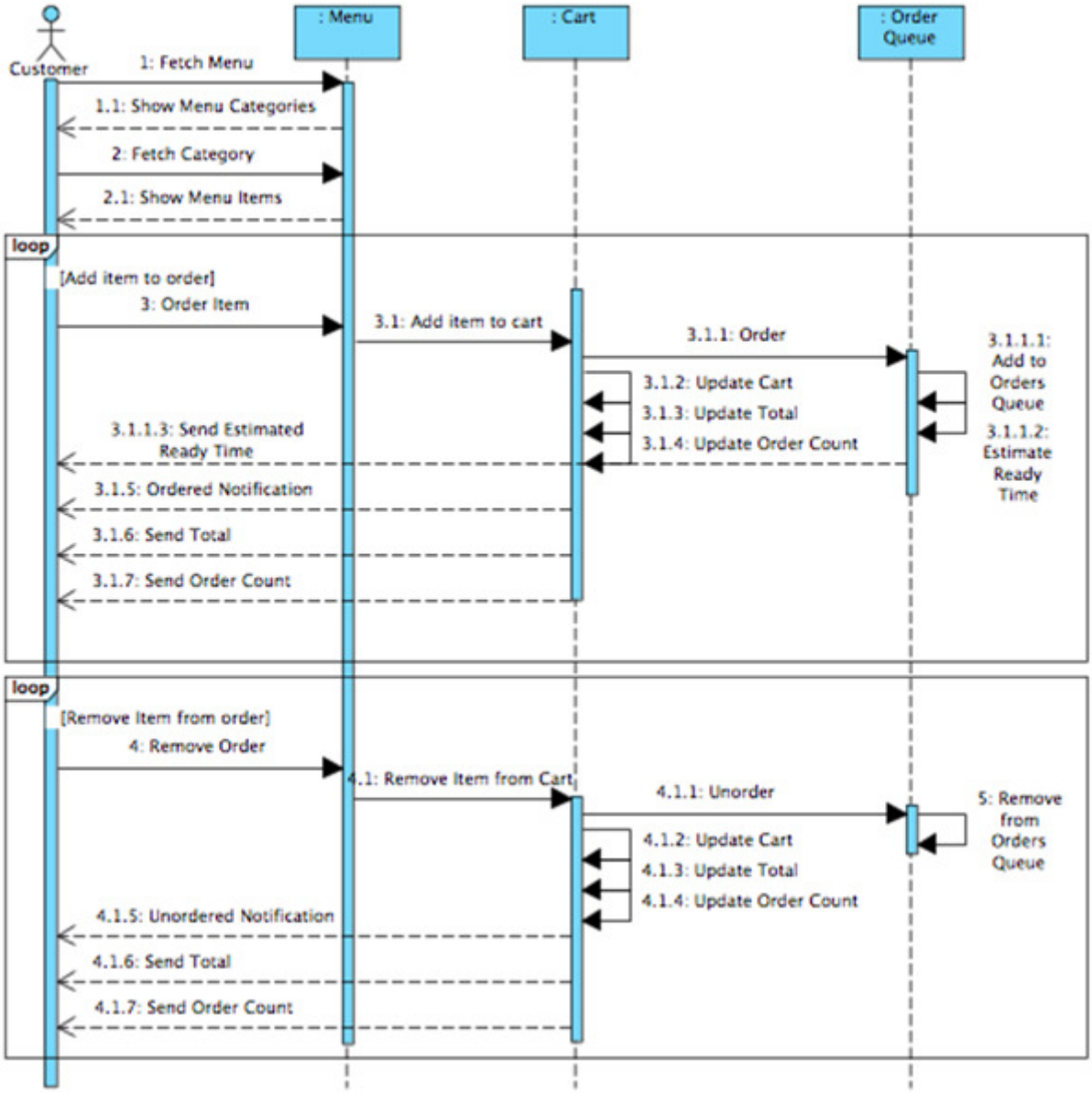
→ 4b. Bartender or manager selects "Change Item" next to the item he/she wants to change.
← System displays pop-up window with some textboxes to change "Name:", "Price:", and "Inventory Count:" of the existing item.

→ Bartender or manager enters information and hits the "Update Item" when finished.

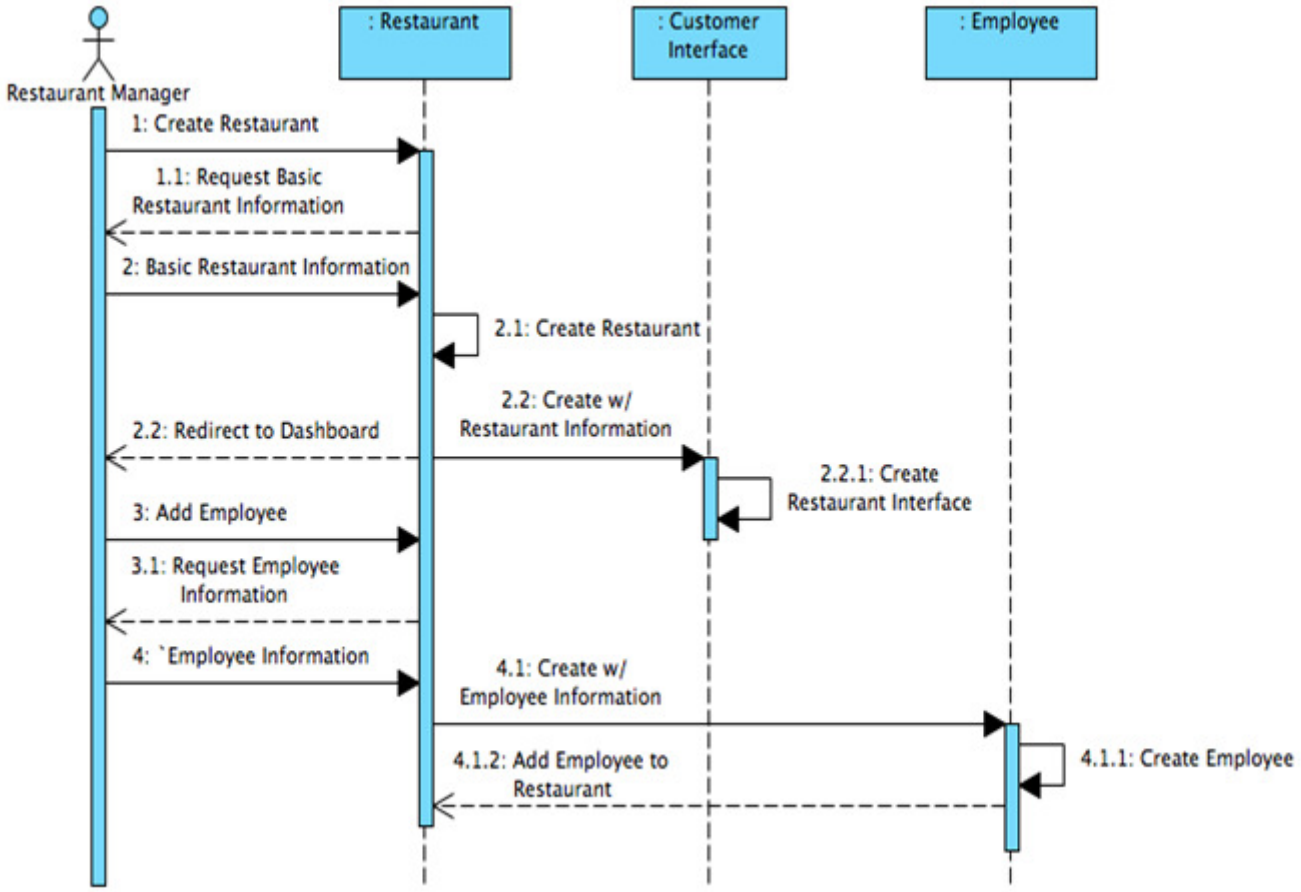
← System updates the item and returns to the items list.

D. System Sequence Diagrams:

Place Order as a Restaurant Customer:

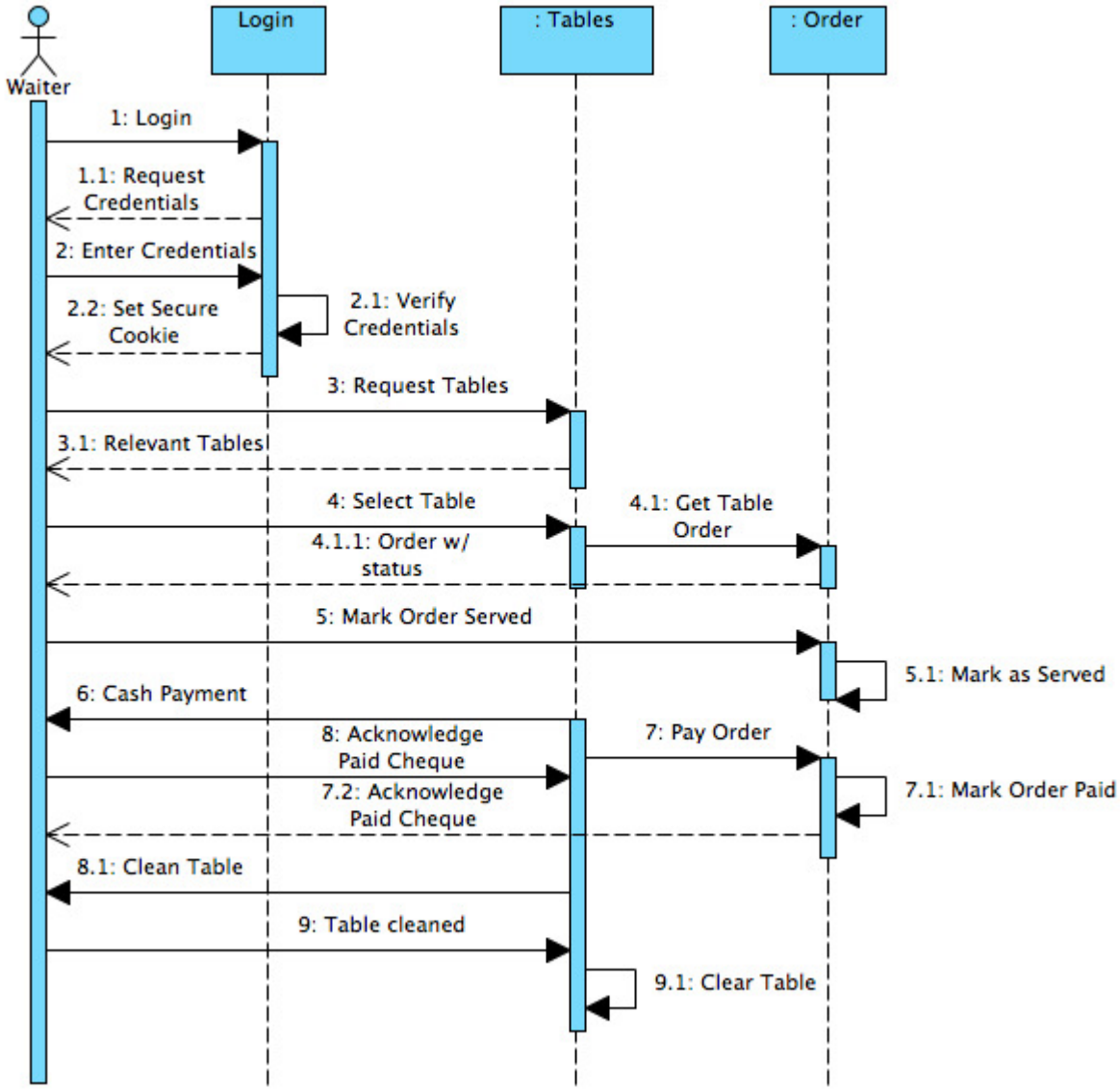


Create a Restaurant and Add an Employee as a Manager:

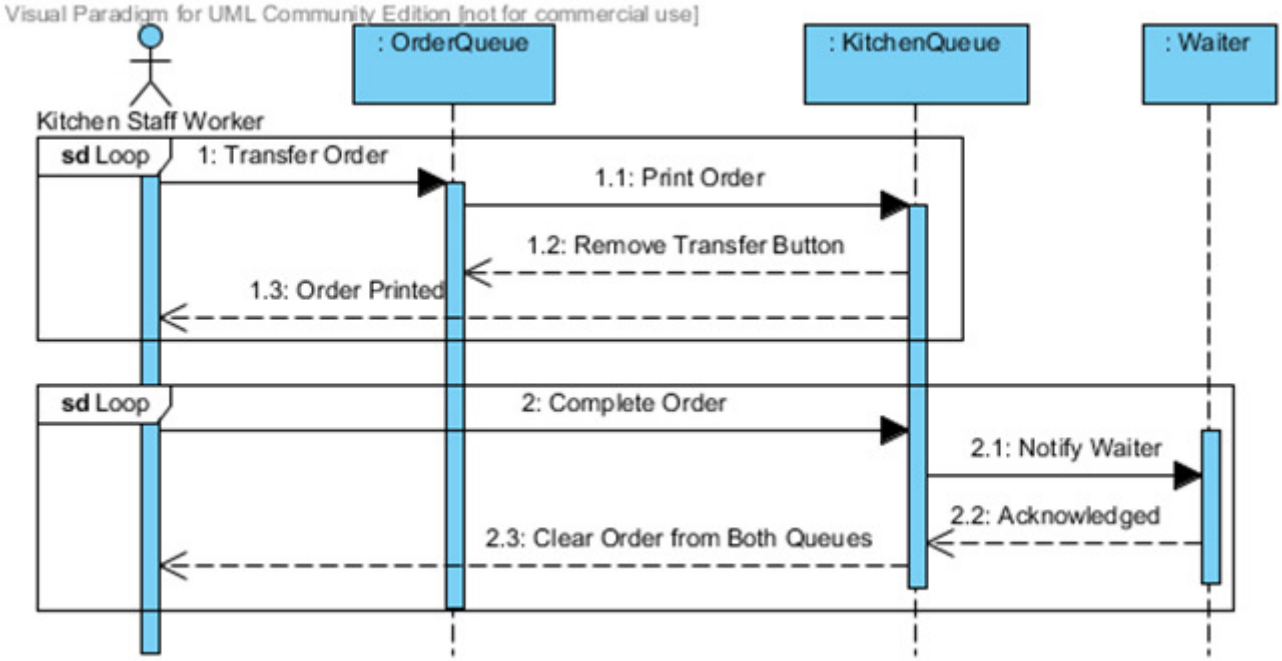


The manager of the restaurant should be able to create a webpage that shows the basic restaurant information. A customer interface should also be created from that restaurant subdomain within GravyXpress. Furthermore, a manager should be able to add an employee to the restaurant subdomain by sending a request along with the basic information.

Serve Table as Waiter:

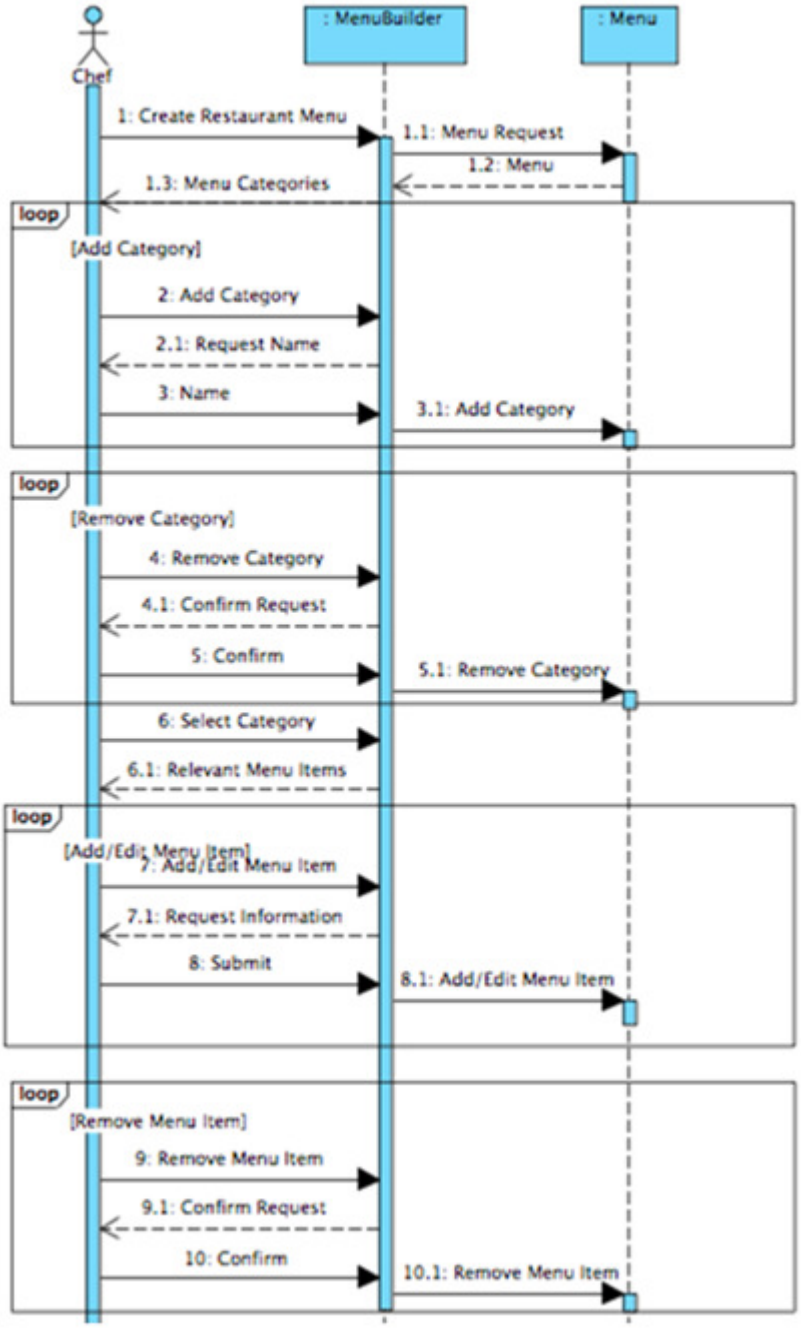


Prepare Order as a Kitchen Staff Worker:



This sequence diagram was updated from Report 1 according to how it was actually implemented. We have added a Waiter lifeline to show how the kitchen staff worker interacts with the waiter. We have also minimized button presses by allowing the kitchen staff worker to view both the Order Queue and Kitchen Queue on the same interface. Furthermore, we detected and eliminated a common user error (possible multiple clicks on the same order to transfer to Kitchen Queue), by telling the system to "Remove Transfer Button" once the order has been moved to the Kitchen Queue.

Manage Menu as a Chef:



This use case needs not to be revised, it pretty straight forward. The main idea of this use case is to let chef change the menu according to the needs of the restaurant. If ingredients will run out chef will have ease to remove the items from the menu also chef can add more items once they're available to sell.

E. Traceability Matrix Mapping Use Cases with User Stories:

User Stories	UC-1	UC-2	UC-3	UC-4	UC-5
ST-C-1	X				
ST-C-2		X			
ST-C-3	X				
ST-C-4		X			
ST-M-1		X			
ST-M-4		X			
ST-M-5		X			
ST-M-6		X			
ST-W-1			X		
ST-W-2			X		
ST-W-3			X		
ST-W-4			X		
ST-K-1				X	
ST-K-3				X	
ST-K-5				X	
ST-K-6				X	
ST-Ch-2					X
ST-Ch-3					X
ST-Ch-4					X
ST-Ch-5					X

5. Effort Estimation using Use Case Points:

1 **ManageOrder (Kitchen Staff Worker) UC-4:**
Unadjusted Actor Weight:

Actor Name	Description of relevant characteristics	Complexity	Weight
Kitchen Staff Worker	Kitchen staff worker is interacting with the system via graphical user interface.	Complex	3
Customer	Customer is sending orders via graphical user interface.	Complex	3
Database	Subsystem acting through a protocol.	Average	2
Waiter	Receives text notifications of orders that are ready.	Simple	1
Chef	Enables or disables items from the menu when ingredients have run out as well as add more items to the menu once available to sell via graphical user interface.	Complex	3

UAW = 3 x 3 + 2 + 1 = 12 points

Unadjusted Use Case Weight:
There are a total of 6 steps in the main success scenario. There are three participating actors, where one is offstage. And there are a total of 5 concepts in this use case. Hence, UC-4 has a moderate interface design and it's use case category is Average with a weight of 10 points. Therefore, the UUCW for UC-4 is 10 points.

Unadjusted Use Case Points:
UUCP = UAW + UUCW = 12 + 10 = 22 points

Technical Complexity Factor:

Technical factor	Description	Weight	Perceived Complexity	Calculated Factor
T1	May or may not run on multiple machines. Can be accessed on web.	2	1	2
T2	Both queues must be updated regularly, so that kitchen staff can see customer orders and prepare food.	1	3	3
T3	Kitchen staff expect good efficiency, but no exceptional demands.	1	3	3
T4	Internal processing is there, but is simple.	1	1	1
T5	There will be no other module reusing the code for the Kitchen Queue.	1	0	0

T6	No installation required. This is part of a web app.	0.5	0	0
T7	Usage is important and it should be easy to use.	0.5	5	2.5
T8	It is portable as it is on the web. No concerns.	2	0	0
T9	It is easy to change. Minimal change is required.	1	1	1
T10	Concurrent usage is not required. Only one kitchen staff member will be seeing this.	1	0	0
T11	Security is important. No other actor or customer should access the Kitchen interface.	1	4	4
T12	No access to third parties.	1	0	0
T13	Minimal user training needed.	1	2	2
	Technical Factor Total:			**18.5**

TCF for UC-4 = 0.6 + 0.01 x 18.5 = 0.785

Environment Complexity Factor:

Environmental factor	Description	Weight	Perceived Complexity	Calculated Factor
E1	Beginner development of the Kitchen Interface using UML diagramming.	1.5	1	1.5
E2	Average familiarity with Kitchen Interface problems.	0.5	3	1.5
E3	Average knowledge of Object-Oriented approach.	1	3	3
E4	Beginner lead analyst.	0.5	1	0.5
E5	Highly motivated to develop the Kitchen Interface.	1	4	4
E6	Some stable and some unstable requirements.	2	3	6
E7	Average half of the team is part time to develop the Kitchen Interface.	-1	3	-3
E8	We use a fairly difficult programming environment (Play Framework) to develop the Kitchen Interface.	-1	4	-4
	Environmental Factor Total:			**9.5**

ECF for UC-4 = 1.4 - 0.03 x 9.5 = 1.115

Total Use Case Points for UC-4:
UCP = UUCP x TCF x ECF = 22 x 0.785 x 1.115 = 19.256 points

Effort Estimation for UC-4:

Duration = UCP x PF = 19.256 x 28 = 539 hours

Therefore, Use Case 4 (UC-4) ManageOrder took approximately 539 hours to design and implement.

2 **Create Restaurant (Kitchen Staff Worker) UC-2:**

Unadjusted Actor Weight:

Actor Name	Description of relevant characteristics	Complexity	Weight
Restaurant Manager	Adds new restauarant to database via GUI.	Complex	3
Restaurant Manager	Modifies restaurant preferencess via GUI dashboard.	Complex	3
Database	Subsystem acting through a protocol.	Average	2
General User	Visits website's page which is rendered with a GUI.	Complex	3
Controller	Handles HTTP requests	Average	2

UAW = 3 x 3 + 2 = 11 points

Unadjusted Use Case Weight:

There are a total of 6 steps in the main success scenario. There are two participating actors, where one is offstage. And there are a total of 5 concepts in this use case. Hence, UC-4 has a moderate interface design and it's use case category is Average with a weight of 10 points. Therefore, the UUCW for UC-4 is 10 points.

Unadjusted Use Case Points:

UUCP = UAW + UUCW = 11 + 10 = 21 points

Technical Complexity Factor:

Technical factor	Description	Weight	Perceived Complexity	Calculated Factor
T1	May or may not run on multiple machines. Can be accessed on web.	2	1	2
T2	Passwords are all hashed using a secure algorithm before storage.	1	2	2
T3	Dashboards should be forbidden to users who are not logged in as owners.	1	3	3
T4	Internal processing creates blank menu.	1	1	1
T5	There will be no other module reusing the code for the Kitchen Queue.	1	0	0
T6	Software is installed and hosted on server alone.	0.5	0	0
T7	Usage is important and it should be easy to use.	0.5	5	2.5

T8	It is portable as it is on the web. No concerns.	2	0	0
T9	It is easy to change. Minimal change is required.	1	1	1
T10	Concurrent usage is required. Multiple restaurant owners should be able to create restaurant's simultaneously	1	1	1
T11	Security is extremely important. Hackers shouldn't be able to access any restaurant owners dashboards or alter settings.	1	4	4
T12	No access to third parties.	1	0	0
T13	Minimal user training needed.	1	2	2
	Technical Factor Total:			**18.5**

TCF for UC-2 = 0.6 + 0.01 x 18.5 = 0.785

Environment Complexity Factor:

Environmental factor	Description	Weight	Perceived Complexity	Calculated Factor
E1	Beginner development of the Restaurant Creation using UML diagramming.	1.5	1	1.5
E2	Average familiarity with Restaurant Creation problems.	0.5	3	1.5
E3	Average knowledge of Object-Oriented approach.	1	3	3
E4	Beginner lead analyst.	0.5	1	0.5
E5	Highly motivated to develop the Restaurant Creation Interface	1	4	4
E6	Some stable and some unstable requirements.	2	3	6
E7	Average half of the team is part time to develop the Kitchen Interface.	-1	3	-3
E8	We use a fairly difficult programming environment (Play Framework) to develop the Kitchen Interface.	-1	4	-4
	Environmental Factor Total:			**9.5**

ECF for UC-2 = 1.4 - 0.03 x 9.5 = 1.115

Total Use Case Points for UC-2:
UCP = UUCP x TCF x ECF = 21 x 0.785 x 1.115 = 18.38 points

Effort Estimation for UC-2:
Duration = UCP x PF = 18.38 x 28 = 514 hours

Therefore, Use Case 2 (UC-4) ManageOrder took approximately 514 hours to design and implement.

3 **Serve table (Waiter) UC-3:**

Unadjusted Actor Weight:

Actor Name	Description of relevant characteristics	Complexity	Weight
Restaurant Manager	Upload the tables to the restaurant database	Simple	1
Waiter	Views and serves the tables which have been assigned	Complex	3
Database	Subsystem acting through a protocol.	Average	2
Customer	Order specific items from the menu	Complex	3
Controller	Handles HTTP requests	Average	2

UAW = 1+3+2+3+2 = 11 points

Unadjusted Use Case Weight:

There are a total of 12 steps in the main success scenario. There are two main participating actors. And there are a total of 5 concepts in this use case. Hence, UC-3 has a moderate interface design and it's use case category is Average with a weight of 10 points. Therefore, the UUCW for UC-4 is 10 points.

Unadjusted Use Case Points:

UUCP = UAW + UUCW = 11 + 10 = 21 points

Technical Complexity Factor:

Technical factor	Description	Weight	Perceived Complexity	Calculated Factor
T1	Will not run on multiple machines. Can be accessed on web.	2	1	2
T2	User accounts for each user are stored in the database	1	1	1
T3	Waiters only see the tables that they are assigned to	1	3	3
T4	Internal processing creates blank menu.	1	1	1
T5	Software is installed and hosted on server alone.	0.5	0	0
T6	Usage is important and it should be easy to use.	0.5	5	2.5
T7	It is portable as it is on the web. No concerns.	2	0	0
T8	It is easy to change. Minimal change is required.	1	1	1

T9	Concurrent usage is required. Multiple restaurant owners should be able to create restaurant's simultaneously	1	1	1
T10	No access to third parties.	1	0	0
T11	Employees will be trained for their own usage of the software.	1	2	2
	Technical Factor Total:			**13.5**

TCF for UC-3 = 0.6 + 0.01 x 13.5 = 0.735

Environment Complexity Factor:

Environmental factor	Description	Weight	Perceived Complexity	Calculated Factor
E1	Very little prior experience of UML diagrams, but effective class lectures on the subject	1.5	1.5	2.25
E2	Little familiarity with table serving problems.	0.5	3	1.5
E3	Average knowledge of Object-Oriented approach.	1	3	3
E4	Beginner lead analyst.	0.5	1	0.5
E5	Highly motivated to develop the Serve table Interface	1	4	4
E6	Some stable and some unstable requirements.	2	3	6
E7	Using a lower level and perhaps more involved framework, than perhaps doing something akin to PHP or Python.	-1	4	-4
	Environmental Factor Total:			**13.25**

ECF for UC-2 = 1.4 - 0.03 x 13.25 = 1.0025

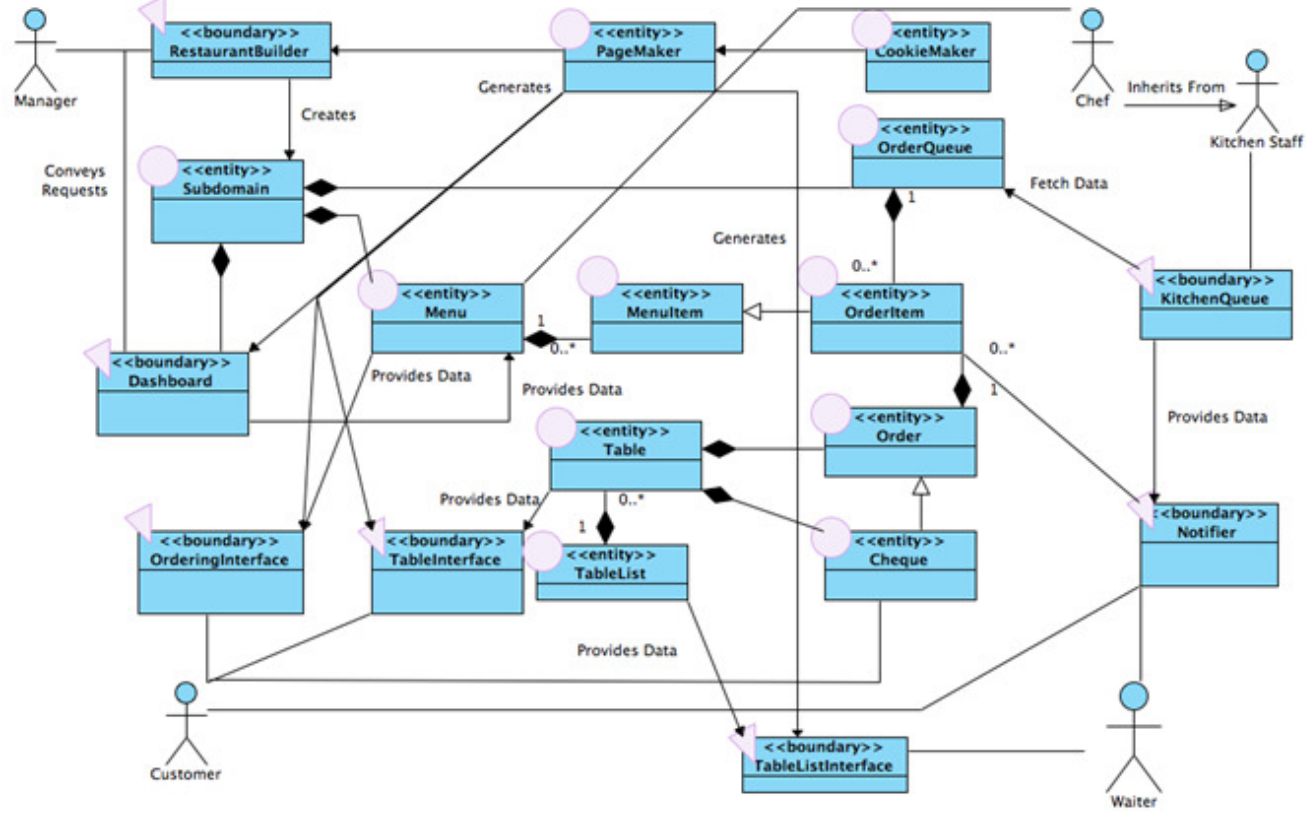
Total Use Case Points for UC-2:
UCP = UUCP x TCF x ECF = 21 x 0.735 x 1.0025 = 15.47 points

Effort Estimation for UC-2:
Duration = UCP x PF = 15.47 x 28 = 433 hours

Therefore, Use Case 3 (UC-3) Serve tables took approximately 433 hours to design and implement.

6. Domain Analysis:

A) Domain Model:



Although the scope of this project extends that of the Domain Model drawn above, the above represents the core of GravyXpress as we envision it. We view GravyXpress as a cloud based communication system that facilitates communication between Customers, Managers, Waiters, Chefs, and Kitchen staff. Any other feature is secondary.

In this simplified representation, in order to keep the relationships between the different objects and players clear, we have eliminated some of the objects that clutter the domain model. One such object is a controller to interface between other objects in the domain. Such an object makes it difficult to trace communication, and will only be included in the design phase when we outline the attributes and methods of individual objects.

Objects with circles at their left upper corner represent entities, while those with triangles represent boundary cases.
A summary of the domain model is as follows:

1 Upon visiting any page in the GravyXpress website, the page-maker renders all of the html dynamically, if necessary setting secure cookies with the cookie maker.

2 A restaurant manager creates a restaurant through the restaurant creation module.
a) In doing so, he initializes the restaurants menu, capacity, order queue, and kitchen queue to empty or default values.

3 Upon placing an order (by either a customer or a waiter) items from the menu are combined with table information as well as other information relevant to an order and passed into the order queue.
4 When the Kitchen Staff is ready to prepare the next order, the staff will fetch the order from the Order Queue into the Kitchen Queue. This Kitchen Queue is the list of items that the kitchen staff will prepare. It is different from the Order Queue because this is a way for the kitchen staffers to pace themselves as they cook.
5 When order items are added or removed from a table's order, this change will feature in the Order Queue.
6 Customers can send real time notifications telling waiters to service their table.
7 GravyXpress also keeps a dynamic cheque that updates itself as more items are added or removed from a customer's order.

In the design of this Domain Model, we have buried several interesting innovations.

The first of these innovations is the realtime notification system that represents the core of GravyXpress's communication infrastructure. Rather than continue polling the server for incoming notifications over HTTP, we will use the the HTML5 websockets two way communication infrastructure to enable the server to notify clients when appropriate rather than simply responding to client HTTP requests.

The reason for implementing GravyXpress's communication system this way is based on the fact that we intend GravyXpress to be a cloud based service. If many restaurants all over the world create subdomains, we don't want clients to pummel GravyXpress's server with HTTP requests continuously. The new HTML5 websockets infrastructure allows connections to remain open and for the server to initialize requests.

Another interesting innovation is the separation between the Kitchen Queue and the Order Queue. The philosophy behind this decision is that we want customers to be able to change their mind about an order right until the kitchen has begun to prepare it.

We therefore require the kitchen staff to retrieve orders from the Order Queue rather than simply pushing all orders directly to them. Aside from this benefit, this design model also ensures that the Kitchen Staff are able to retrieve items whenever is necessary.

The problem was that the kitchen staff had no way of knowing whether the restaurant was busy, and could not pace themselves accordingly. We worked around this problem by designing the Kitchen Queue with an interface that tells the kitchen staff how many orders are in the order queue in real time.

i) Concept Definitions:

Responsibility Description	Type	Concept Name
Rs1. Customer with a unique order id whenever they order something.	K	Customer
Rs2. Customer accesses the menu and places orders.	D	OrderItem
Rs3. Knows all orders from all customers/tables.	K	OrderQueue
Rs4. Knows the status of all tables (Open, Occupied, Needs Cleaning)	K	Table
Rs5. Waiter is chosen and shows the customers at chosen table.	D	SeatCustomer

Rs6. Waiter sees and knows all calls for customer help.	K	SignalWaiter
Rs7. Knows all of the customer's orders and how much they owe.	K	Cheque
Rs8. Kitchen staff sees Order Queue and selects orders to transfer to the Kitchen Queue.	D	TransferOrder
Rs9. Knows all orders selected to prepare for the customers.	K	KitchenQueue
Rs10. Knows when the order is complete and signals waiter for pick up.	K	Complete
Rs11. Knows all possible order items and their prices.	K	Menu
Rs12. Chef or Manager adds or deletes items in the menu.	D	EditMenu
Rs13. Manager creates a subdomain for his/her own restaurant.	K	Restaurant
Rs14. Knows all employees, their positions, and their personal data.	K	Employee
Rs15. Utilizes Play Framework to process all HTTP requests.	D	RequestHandler
Rs16. Creates new interfaces after Rs13 has been completed.	D	PageMaker

ii) Association Definitions:

Concept Pair	Association Description	Association Name
Customer ⇔ OrderItem	Customer orders an item to be placed into the Order Queue.	conveys requests/ requests save
OrderItem ⇔ OrderQueue	Customer's orders are placed into the Order Queue.	provides data
Table ⇔ SeatCustomer	Waiter should be able to change the status of the table in the system after seating the customer.	provides data
Customer ⇔ Cheque	Customer should be able to see what they orders and how much they owe.	provides data
Customer ⇔ SignalWaiter	Customer can signal waiter any time for help.	conveys requests
Restaurant ⇔ PageMaker	The creation of the subdomain should also create all the different interfaces of GravyXpress	prepares
OrderQueue ⇔ KitchenQueue	The OrderQueue provides data for the KitchenQueue so orders can be moved.	provides data
TransferOrder ⇔	Saves orders from the OrderQueue into the KitchenQueue.	requests save

KitchenQueue		
KitchenQueue ⇔ Complete	Notifies waiter that food is ready for pick up.	requests notify
Customer ⇔ Menu	Provides information for the customer about what's on the menu.	provides data
Menu ⇔ EditMenu	Allows manager and chef to edit the menu.	prepares
Restaurant ⇔ Employee	Allows manager to add all employee information to the system.	generates
RequestHandler ⇔ any concept	Processes all requests in the form of HTTP requests.	conveys requests

iii) Attribute Definitions:

Concept	Attributes	Attribute Description
Customer	partySize	Number of customers during visit at restaurant.
	id	Special id number for each order.
	tableNum	Table number where the customer is sitting at.
OrderItem	menuItem	The actual menu item the customer is ordering.
	dateTime	When the order item was placed by customer.
	id	Special id number for each order placed.
	tableNum	Which table the order came from?
OrderQueue	menuItem	The actual menu items.
	id	Special id number for each order.
	tableNum	Which table the order came from?
	size	Number of items in the queue.
Table	tableNum	The set defined number of the table.
	status	The status of table (Open, Occupied, Needs Cleaning).
SeatCustomer	tableNum	The table number that customer is to sit at.
SignalWaiter	tableNum	The table number from which the signal is coming from.
Cheque	menuItem	The items the customer ordered.
	id	The order identification numbers.

	gratuity	The tip the customer can give to restaurant.
	paid	Status (paid or unpaid)
TransferOrder	menuItem	The item being transferred from the Order Queue to the Kitchen Queue.
KitchenQueue	menuItem	The item taken from OrderQueue.
	size	Number of items in the queue.
Complete	id	The order that was completed.
	menuItem	The actual item that was prepared.
	tableNum	Notify waiter where the prepared item must go.
Menu	menuCategories	The actual categories in the menu (e.g. Breads, Appetizers, Drinks, etc.)
	size	Number of items in each category.
EditMenu	menuCategories	The actual categories in the menu (e.g. Breads, Appetizers, Drinks, etc.)
	menuItem	The items in each category.
	changePrice	Change the price of the item.
	add	Add new category or menu item.
	delete	Delete category or menu item.
Restaurant	name	The name of the restaurant.
	tables	Add virtual tables with table numbers to the restaurant.
	contactDetails	Contact information of the restaurant.
	employee	Names of all employees.
	owner	Name of the owner or manager of the restaurant.
Employee	name	Name of employee.
	contactDetails	Home address, phone number, etc.
	position	What is their position in the restaurant.
RequestHandler	get	Handle HTTP GET requests.
	post	Handle HTTP POST requests.
PageMaker	map	Map out the interfaces once subdomain has been created.

iv) Traceability Matrix:

	UC-1	UC-2	UC-3	UC-4	UC-5
Customer	X				
OrderItem	X				
OrderQueue	X			X	
Table			X		
SeatCustomer			X		
SignalWaiter	X		X		
Cheque	X		X		
TransferOrder				X	
KitchenQueue				X	
Complete			X	X	
Menu					X
EditMenu					X
Restaurant		X			
Employee		X			
RequestHandler	X	X	X	X	X
PageMaker		X			

B) System Operation Contracts:

Name:	Place Order
Responsibilities:	Obtain orders from the customer users and add them to the order queue.
Use Cases:	ST-C-1
Exceptions:	None
Preconditions:	The customer must have an account on GravyXpress and be logged in.
Postconditions:	The order is placed in the order queue.

Name:	Create Restaurant
Responsibilities:	Create a restaurant on the GravyXpress domain.
Use Cases:	ST-M-1
Exceptions:	None
Preconditions:	That the restaurant to be created does not have the same name as one already in the GravyXpress databases.
Postconditions:	The restaurant is created.

Name:	Serve Table
Responsibilities:	Tend to the requests of a customer at a given table.
Use Cases:	ST-W-1, ST-W-4
Exceptions:	None
Preconditions:	That a table of customers be ready to order.
Postconditions:	The customer's requests are served and transactions are complete.

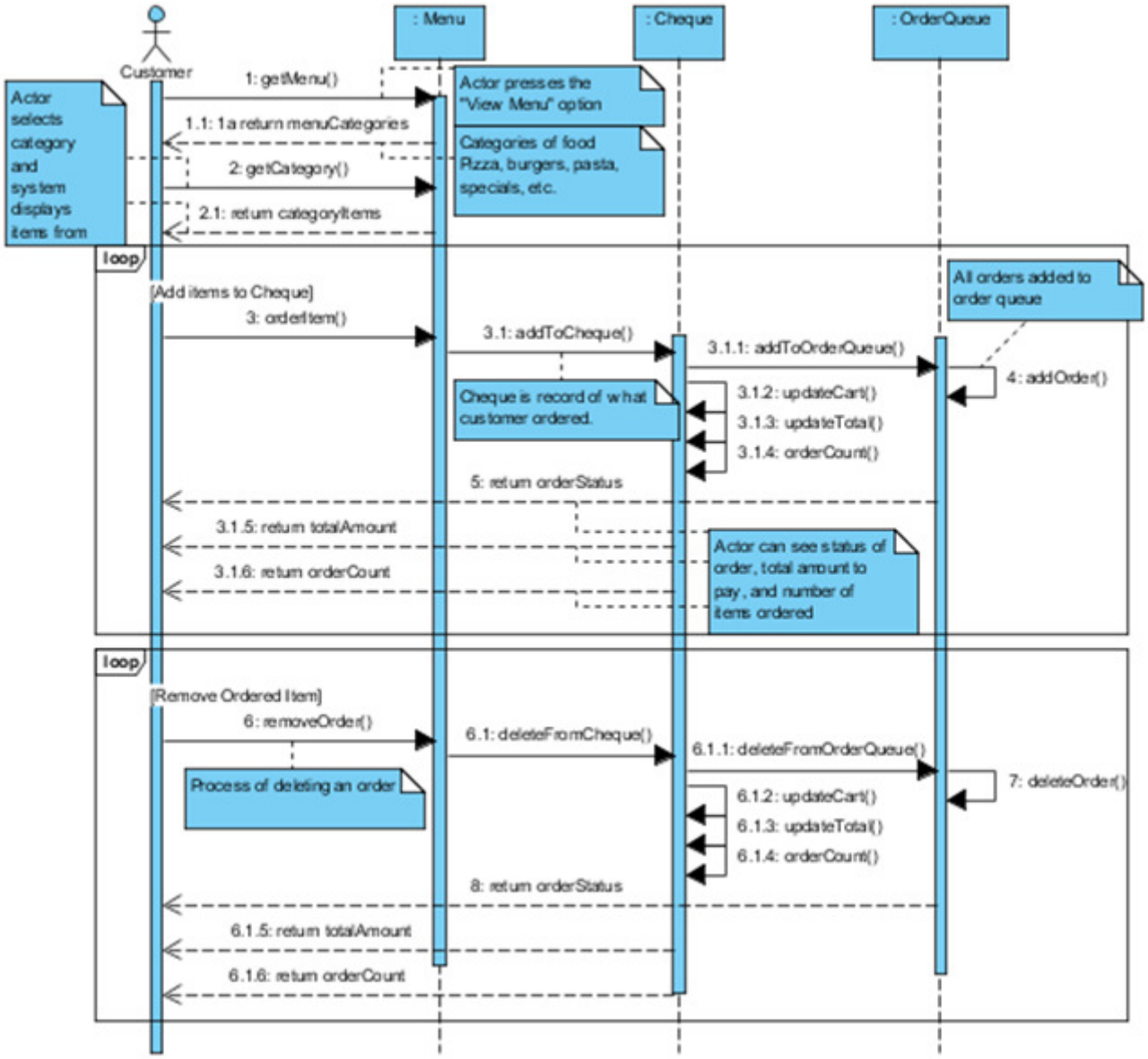
Name:	Prepare Order
Responsibilities:	Obtain orders from the order queue and notify waiter once completed.
Use Cases:	ST-K-1, ST-K-3, ST-K-5
Exceptions:	None
Preconditions:	That an order be on the order queue
Postconditions:	The order has been prepared, and the waiter is notified.

Name:	Manage Menu
Responsibilities:	Update the selection of the menu as a chef.
Use Cases:	ST-Ch-2, ST-Ch-3
Exceptions:	None
Preconditions:	A chef be logged in to the GravyXpress domain for his particular restaurant.
Postconditions:	The menu is updated with the chef's choices.

Name:	About Restaurant
Responsibilities:	Attractive website
Use Cases:	ST-V-1, ST-V-2
Exceptions:	None
Preconditions:	An attractive restaurant website to draw user's attention
Postconditions:	Display the detail about the restaurant

7. Interaction Diagrams:

OrderFood Use Case:



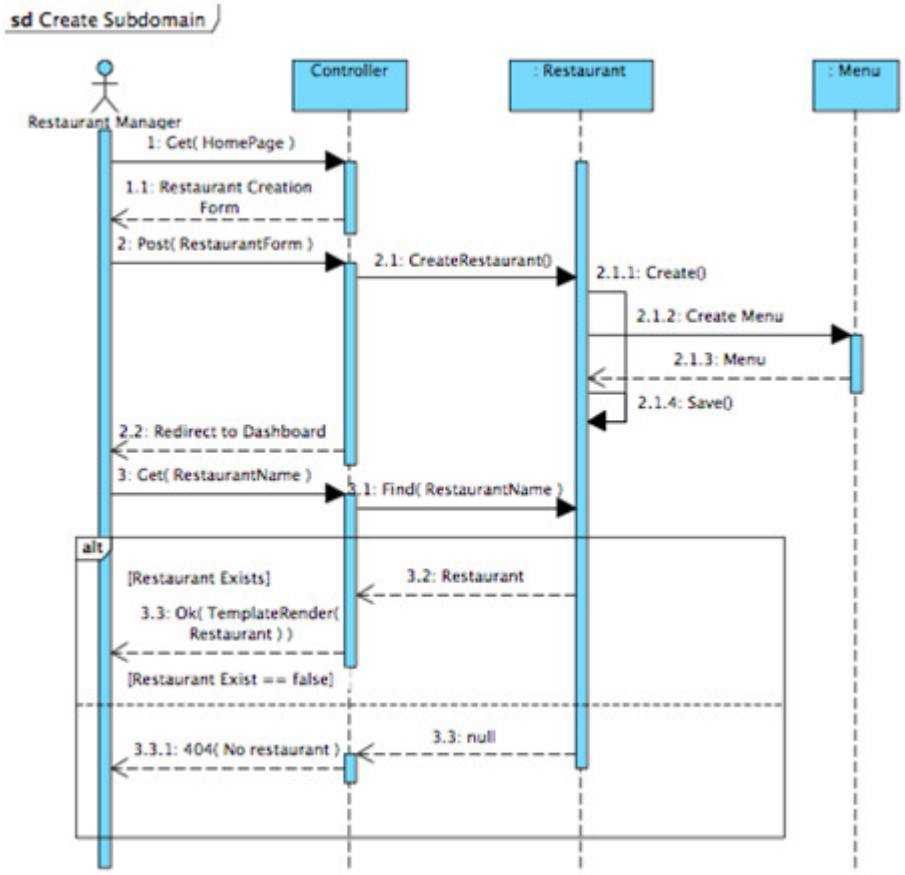
The customer can request the Menu class to see the menu items (organized by categories). It can then send a request to the Menu class to order an item. The Menu class will then send a request to the Cheque class to add the item to the order cheque for the customer. Furthermore, the item will be added 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:



Upon arriving at the GravyXpress home page, a manager is prompted to add his restaurant to GravyXpress. The restaurant's name, a username for the manager and a password field are visible in the signup interface. Each of these fields is required for a restaurant to be created.

Upon submitting the restaurant creation form, the form is sent to the server, and the relevant HTTP request handler within the controller class creates a new restaurant object, with default About and Contact parameters. This object is then persisted to the database.

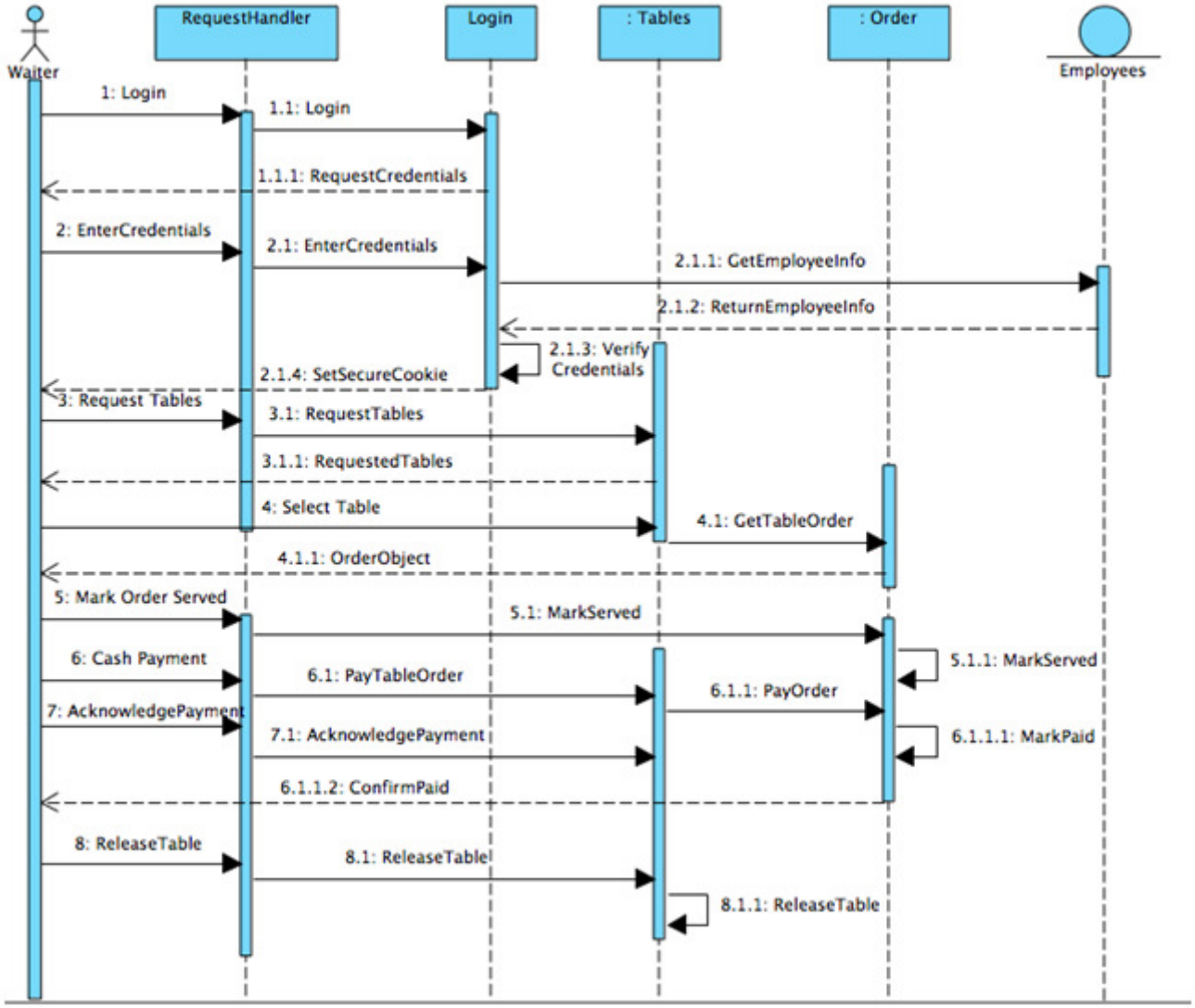
A get request handler handles all requests to render a restaurant's webpage. Upon receiving a get request for a particular restaurant's subdomain, the controller searches the database for a matching restaurant. If the

restaurant is not found, a 404 error is returned. Otherwise, an ok HTTP response is returned, and a restaurant template is rendered with the appropriate customization for the requested restaurant.

This approach of dynamically rendering a single template with customized parameters is far superior to actually creating a separate interface for each restaurant as originally planned, because only a single template needs to be stored rather than an additional template for each restaurant. Additionally, it is far easier to make changes to a restaurant's settings if it is constantly being recreated with every request.

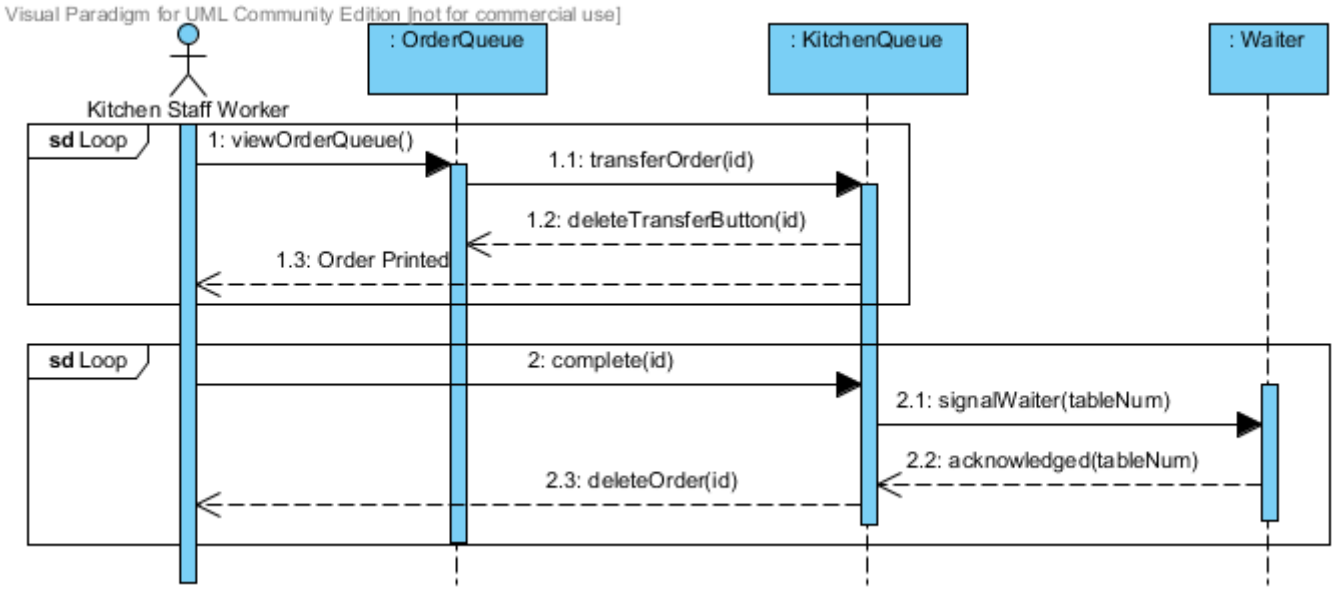
Upon creation of the restaurant, a menu is created. Not shown in the diagram is the propagation of sub-classes that make up a menu. Given that these cascading constructor calls are not key in this user story, they have been omitted to facilitate a cleaner diagram.

ServeTable Use Case:



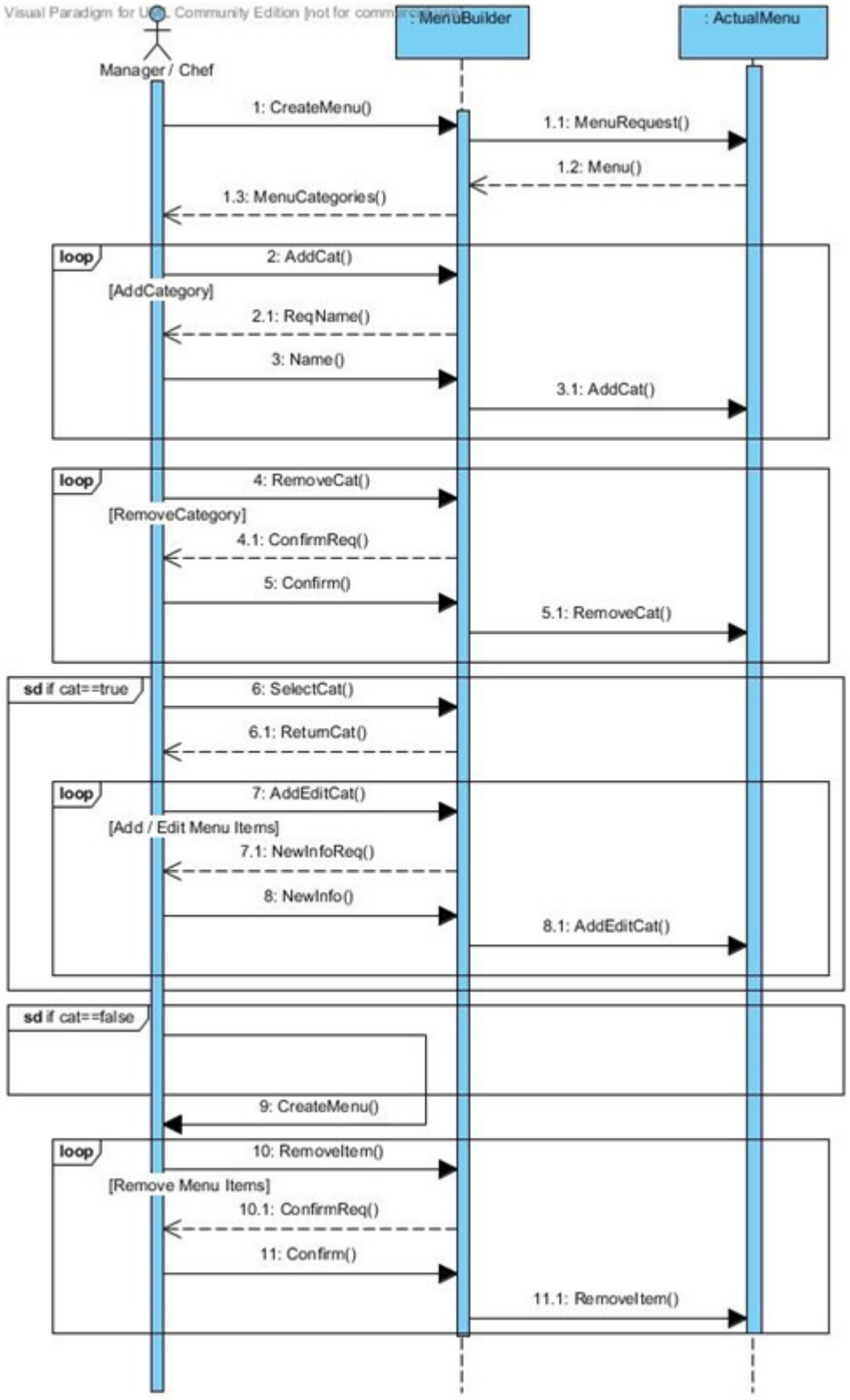
For a waiter to serve a table, he must login, using the login module. Once the login has been verified, he may request his assigned tables. A cookie will be set to ensure that he need not login again until he has closed his browser. He may then select a table from the list, and view the table's order. 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:



This is the interaction diagram for the use case ManageOrder (UC-4). The initiating actor involved is the kitchen staff worker. Basically, the kitchen staff worker views the interface and sees two queues: Order Queue and Kitchen Queue. Orders from customers are coming into the Order Queue. The worker has the liberty to choose which orders to cook first by transferring the orders from the Order Queue to the Kitchen Queue. After the order is prepared, the staffer signals the waiter to pick up the order and deliver to the appropriate table. The order is then deleted from both queues. The whole process repeats again for future orders.

ChangeMenu Use Case:



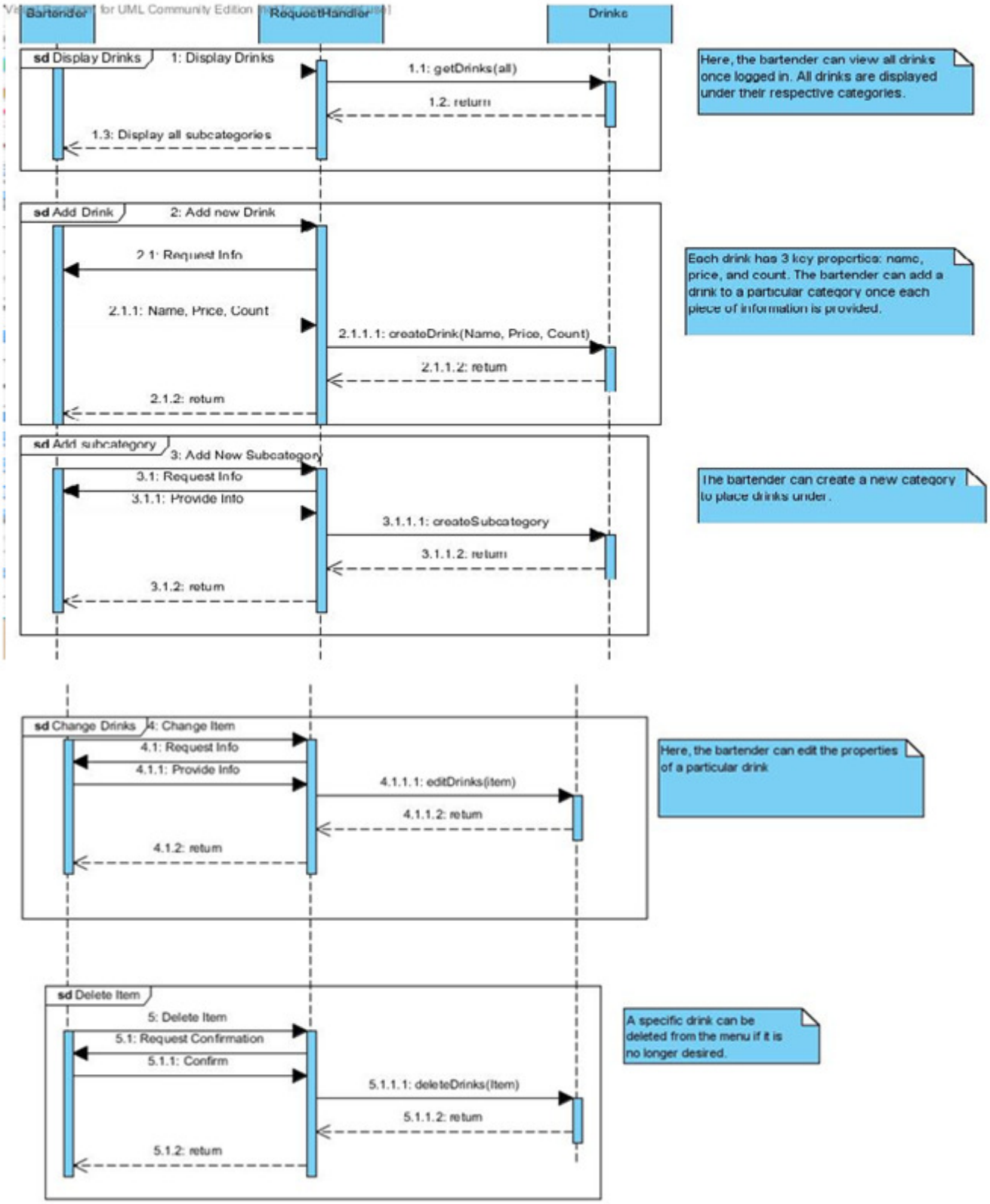
Responsibilities:

- 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**.**

When implementing user stories related to altering the menu, it became apparent that the dashboard was becoming a little cluttered as a result of all of the possible options. As such, for a cleaner design, we opted to remove the option of altering the menu item. Such a feat can be accomplished by simply removing the existing menu item, and creating a new one in its place. The chef can create a new menu item by manually copying the description before removing or disabling the old menu item.

In addition, this approach encourages restaurant owners to simply disable menu items rather than remove them. In subsequent versions of the application, viewing records for each disabled item might be a feature worth implementing.

ChangeDrinks Use Case:



Design Pattern

The primary design pattern we used in implementing this project is the MVC pattern or Models-Views-Controllers pattern. The Play! framework is designed to support this design pattern which is perfect for web applications. Each template is stored as a view, and each class is represented by a model. The controllers handle HTTP requests and render views dynamically using data from the models.
This is the industry standard for web applications, and makes integration a pleasure.

There are many advantages of using the MVC framework. One primary advantage is that it separates the responsibilities of the model, view, and controller classes, making each individual class altogether far more cohesive. Also, it is interesting to note that the views and models act in a kind of publish/subscribe pattern. Whatever the view displays to the user of the web application must reflect what is present in the model, and the model updates the view whenever it is modified. This comes with useful benefits of the publisher/subscriber pattern; namely, the loose coupling between the views and models and improving the scalability of the project as a whole.

Some part of this project also has been implemented in asp.net where the same pattern "MVC" was being used to implement chef stories. Although play framework does the same job, we have decided to use one platform for our integration purposes. Hence asp.net code will be moving from .net to play framework to make integration flawless.

8. 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. But when implemented, it will be shown on a separate interface for testing purposes.

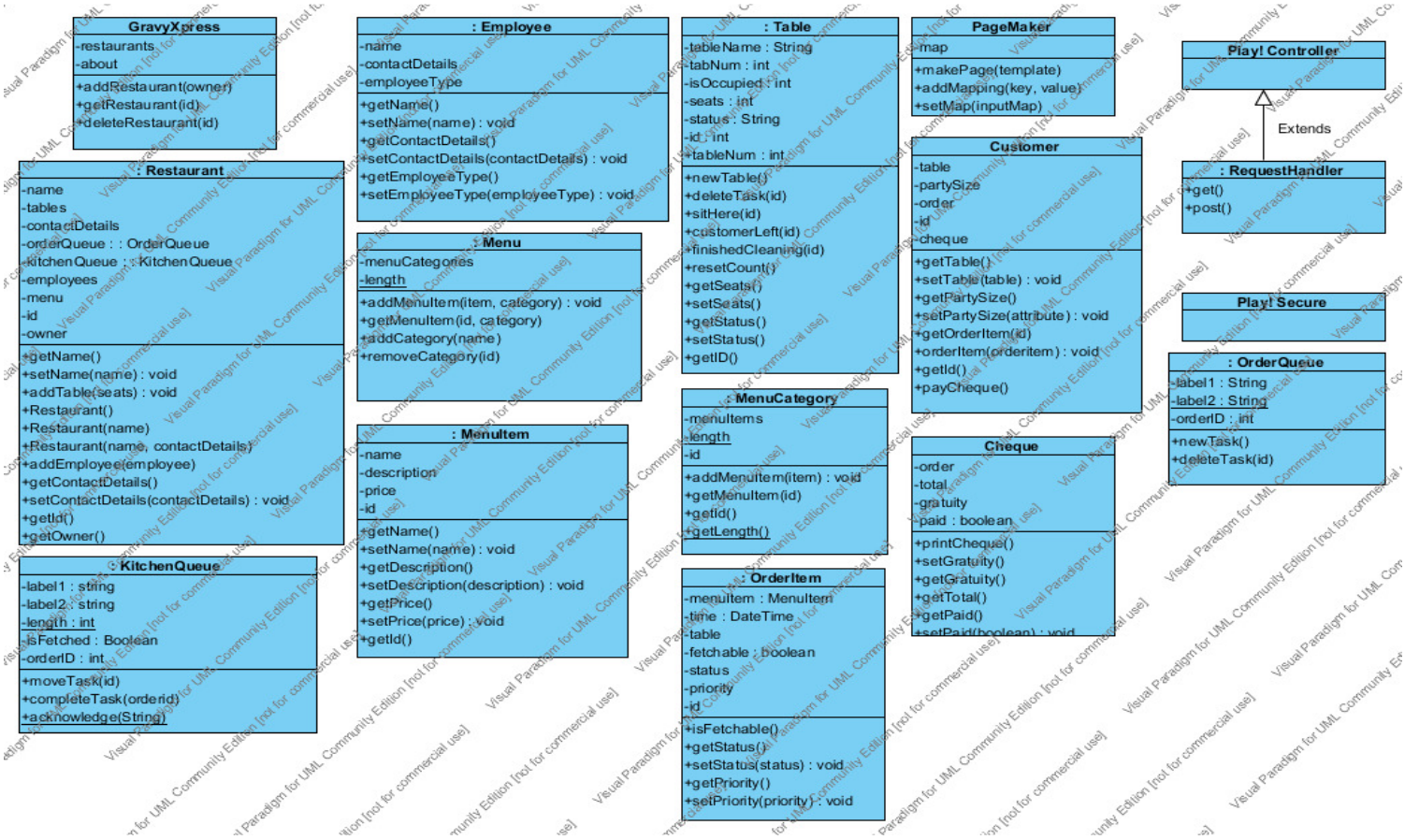
KitchenQueue is a class representing the queue that the Kitchen Staffer sees. Food items will be transferred from the Order Queue to the Kitchen Queue via a press of a button by the Kitchen Staffer.

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:
-label1 : string
// Name of kitchen items.
-label2 : string
// Table number where the orders are coming from. No need to convert to int because no operations will be done with it.
-length : int
// Length of orders
-isFetched:boolean
// Used to decide when to make the "Transfer to Kitchen Queue" button disappear.
-orderID:int
// Randomly generate a unique ID for each order item.
Operations:
+moveTask(id):Result
// This is the method to move the order item from Order Queue to Kitchen Queue. id is a unique database identifier automatically generated by Play's Database.
+compeleteTask(orderId):Result
// This is the method to signal the waiter to pick up completed orders and remove the completed orders from the 2 queues.

+acknowledge(String):Result
// This is the waiter's acknowledge button that will clear his/her list of completed orders after he/she delivers them.

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:

-tableName : String
// Used to hold the string "Table "
-tabNum : int
// The table number assigned to each table created.
-isOccupied : int
// 0, 1, or 2 used to change the -status variable.
-seats : int
// total seats associated with the tables
-status : String
// OPEN, OCCUPIED, NEEDS CLEANING
-id : int
// ID numbers of the seats
+tableNum : static int
// Numbering of the tables.

Operations:

+newTable() : Result
// Creates a new table by calling a model class TableTask.java
+deleteTask(id) : Result
// Deletes a specific table. id is system generated. Special for Play Database.
+sitHere(id) : Result
// Changes -status to OCCUPIED. id is system generated. Special for Play Database.
+customerLeft(id) : Result
// Changes -status to NEEDS CLEANING. id is system generated. Special for Play Database.
+finishedCleaning(id) : Result
// Changes -status to OPEN again. id is system generated. Special for Play Database.
+resetCount() : Result
// Resets table numbering (static variable -tableNum).
+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:
-order : int
-id : int
// Customer's order id
-cheque : double

Operations:+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

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:
-label1 : String
// Holds the order item.
-label2 : String
// Holds the table number. No operations with this variable.
-orderID : int
// Holds the unique ID for each order.

Operations:
+newTask() : Result
// Adds the order item, generates order ID, and adds the table number in to the list
+deleteTask(id) : Result
// Deletes an order. id is system-generated. Special to Play Database.

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).

	Software Classes													
Domain Concepts	GravyXpress	Restaurant	KitchenQueue	Employee	Menu	MenuItem	Table	MenuCategory	OrderItem	PageMaker	Customer	Cheque	RequestHandler	OrderQueue
Customer	X	X							X		X	X		X
OrderItem			X		X	X								X
OrderQueue			X								X			X
Table		X					X					X		
SeatCustomer							X				X			
SignalWaiter											X			
Cheque									X		X			
TransferOrder			X											X
KitchenQueue			X											X
Complete			X				X		X					X
Menu					X	X		X						
EditMenu					X	X		X						
Restaurant	X	X		X						X				
Employee		X		X						X				
RequestHandler	X	X								X			X	
PageMaker	X	X								X			X	

D) Object Constraint Language (OCL) Contracts:

context TablesInterface::AssignWaiter() : Employee
pre: Queue<Employee>: WaiterQueue.get() != NULL
post: Employee: table.Waiter = WaiterQueue.get()

context TablesInterface::AddTable() : TableTask
inv: self.Manager -> True
pre: n/a
post: TableTask: AddTable(int: seats)

context TablesInterFace::DeleteTable() : void
inv: self.Manager -> True
pre: List<TableTask> Tables != NULL
post: void: DeleteTable(Long: id)

context EmployeeForm::AddEmployee() : Employee
inv: self.Manager -> True
pre: n/a
post: Employee: AddEmployee(filledForm.get())

context MenuInterface::AddMenu() : MenuTask
inv: self.Manager -> True
pre: n/a
post: MenuTask: AddMenu(int: item)

context MenuInterFace::UpdateMenu() : void
inv: self.Manager -> True
pre: MenuItem != NULL
post: void: UpdateMenu(int: item)

context MenuInterFace::DeleteMenu() : void
inv: self.Manager -> True
pre: MenuItem != NULL
post: void: DeleteMenu(int: item)

9. System Architecture and System Design:

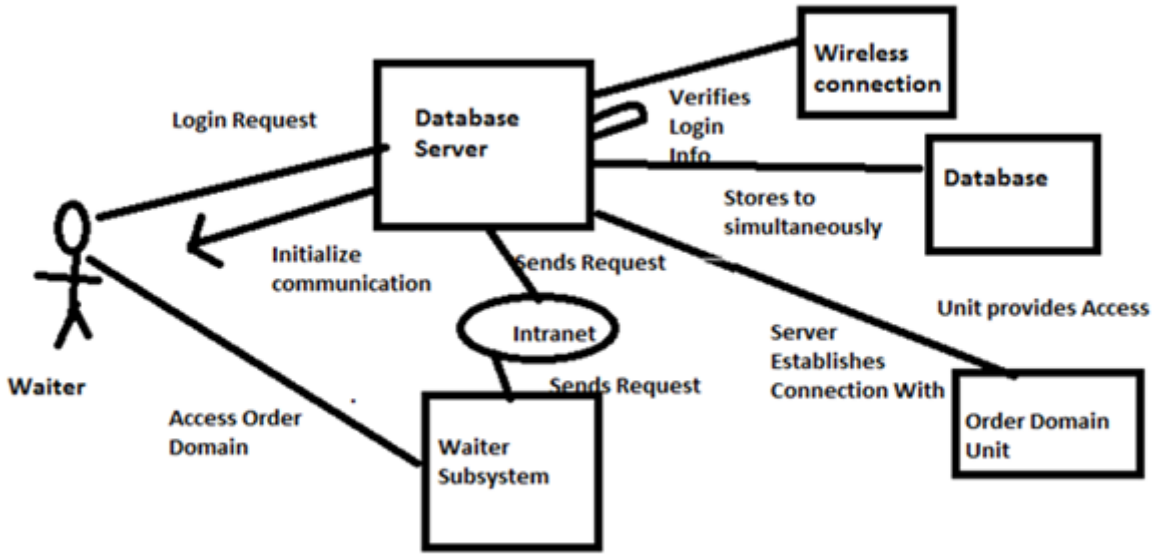
A) Architectural Styles:

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.

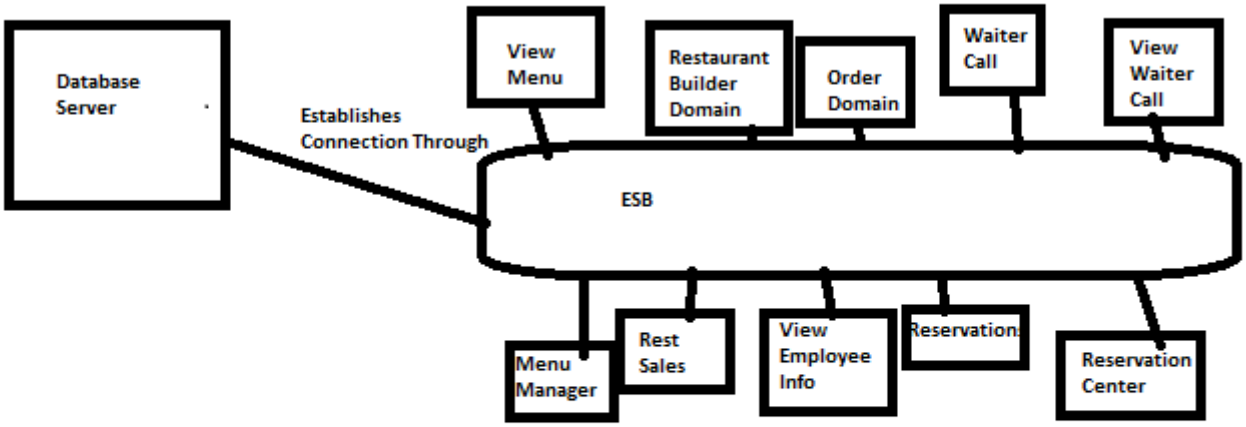
Each user has his or her own terminal that uses a graphical UI to start communication with the database server. The server awaits requests from the clients and then passes along the requests to the correct subdomain. Clients can also communicate with one another through the server. The server also may 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.

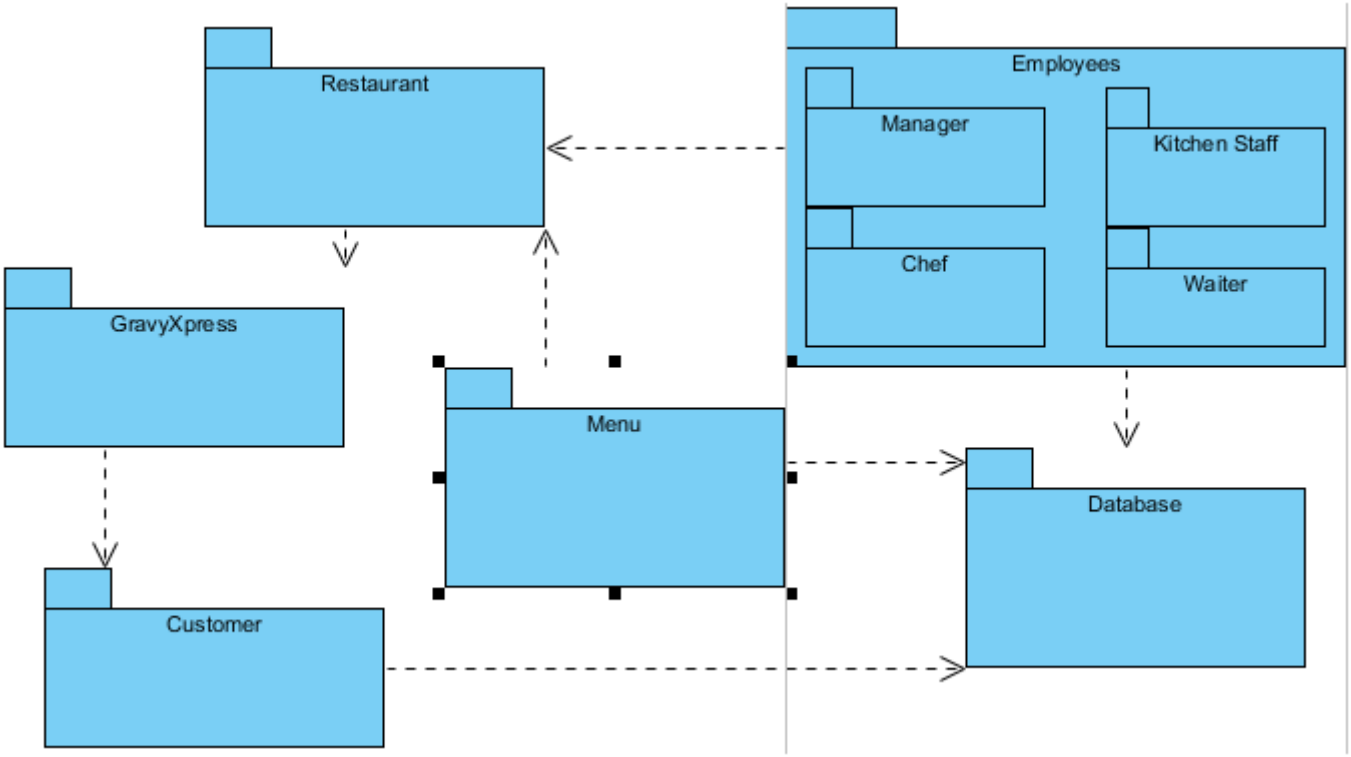


Note: There are other subdomains that were not shown due to lack of space in diagram

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.

List of Subsystems that Fulfill Requirements/User Stories:

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

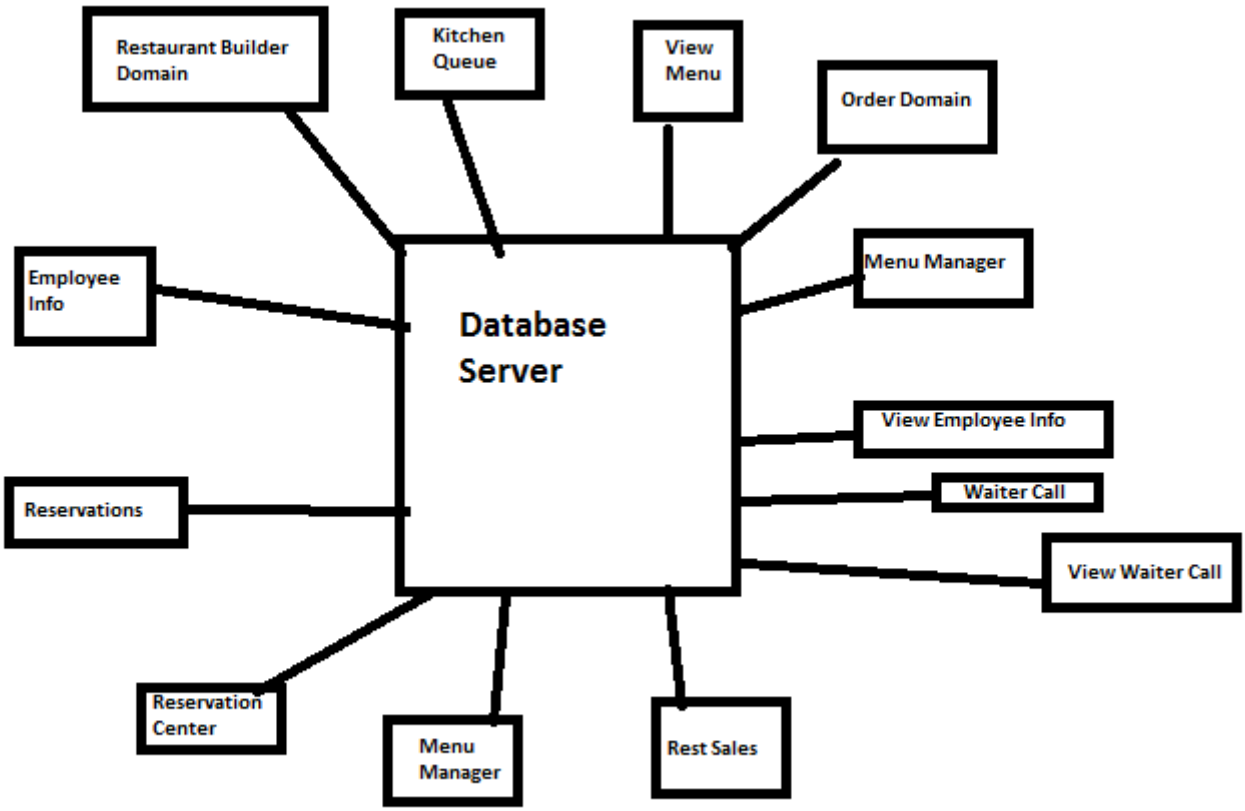
View Reservation - Used by employees to view the reservations

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

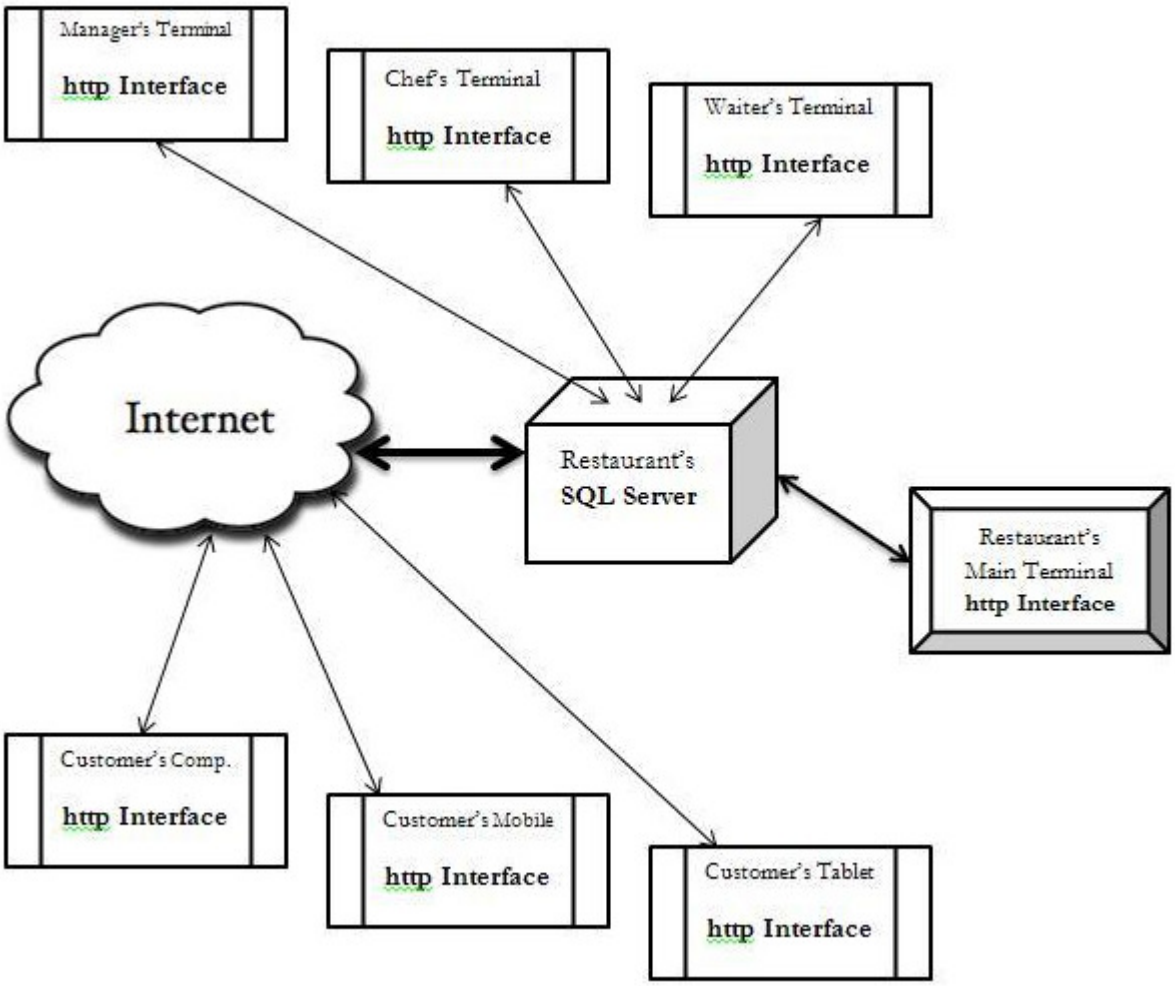
View Table Info - Used by employees to see which tables are available, dirty or clean

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.

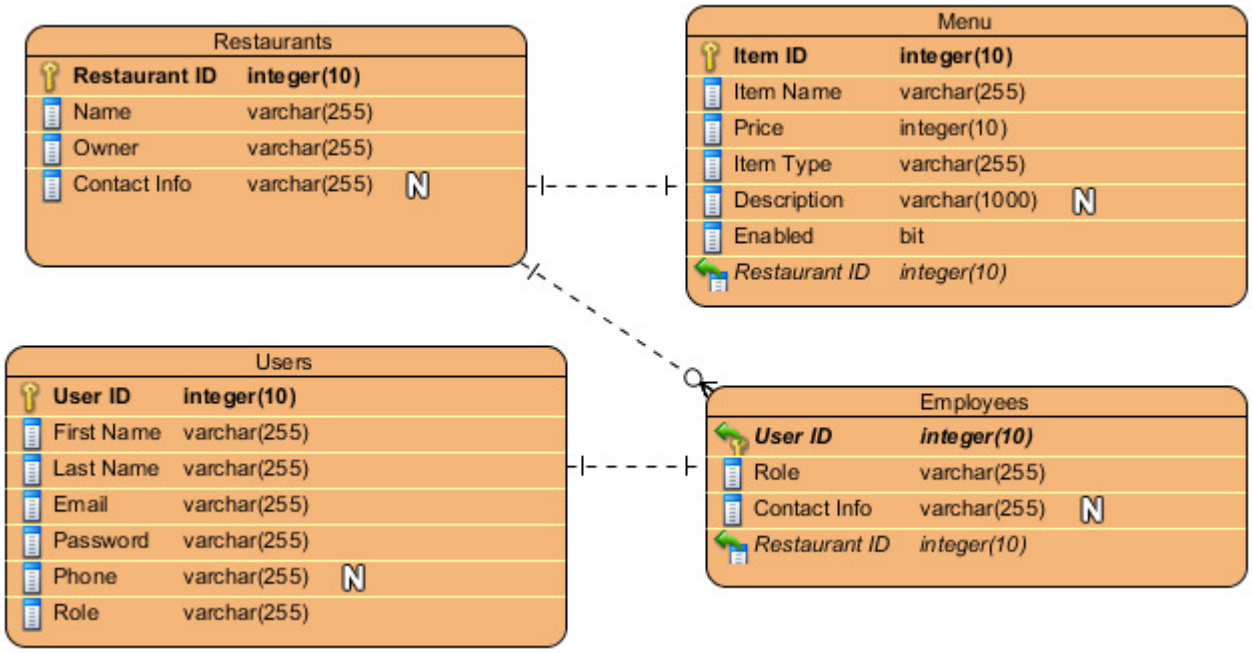
C) Mapping Subsystems to Hardware:



After the completion of our first two reports and first demo, we have decided not to change this part at all. This system pretty much remains the same according to our original plans. Basically that Play! framework that we are using is going to connect everything that is described in above figure.

Play! framework will connect the database with the server where user can only see the front end graphical user interface on their devices at the restaurant. All other http interface will also be connected with the internet that will be handled by controller functions in Play! framework.

D) Persistent Data Storage:



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 are provided for comparison. The Recommended Requirements are the most desirable settings to keep the service running smoothly and leaves room for larger memory requirements.

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 the recommended requirements are provided.

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

10. 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, especially when implementing the Play framework.

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.

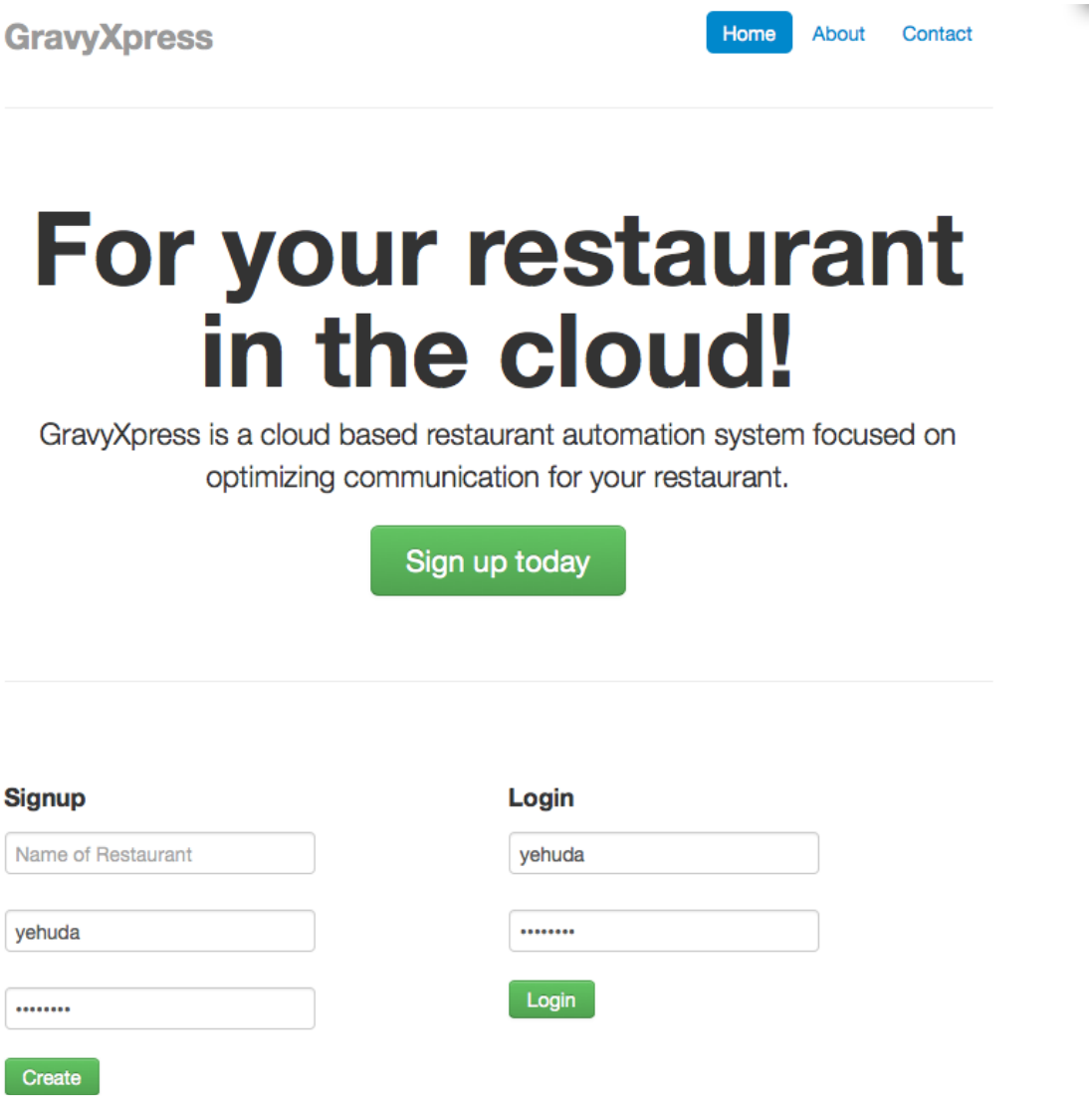
Another instance where a list will be used will be for OrderQueue. The list will also be a priority list since items entered in the list first will be the items that exit the list first. There will be a pointer called "SentToKitchen" which will distinguish between the items sent to the kitchen queue 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.

Furthermore, we will make a linked list of objects that will store the information for each of the order objects. Each order object contains the cost of each item and a list containing information like the table number and waiter name etc.

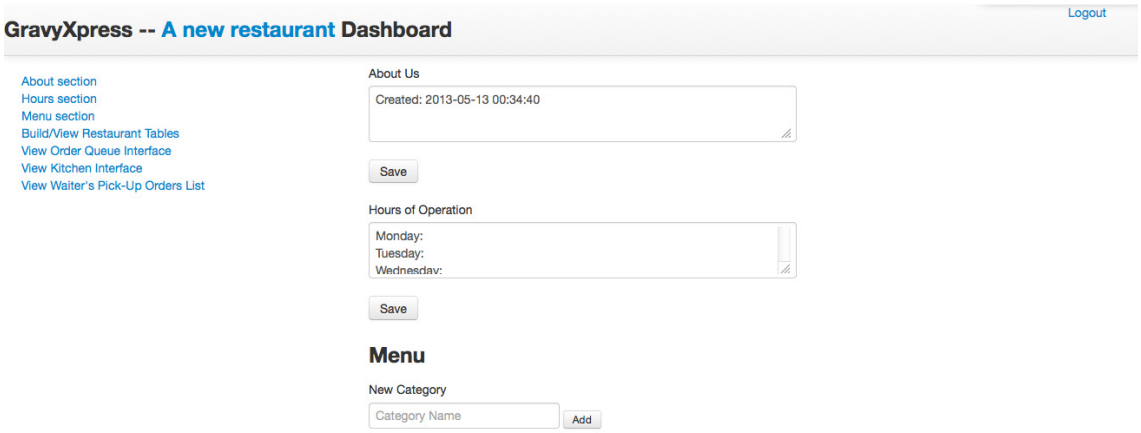
Last but not least, hash tables can be used to map keys to values. This can be used for many aspects of implementation. For instance, a hash table can be used to map the keys (employee names) to their values (payroll amount, telephone number etc.).

11. User Interface Design and Implementation:

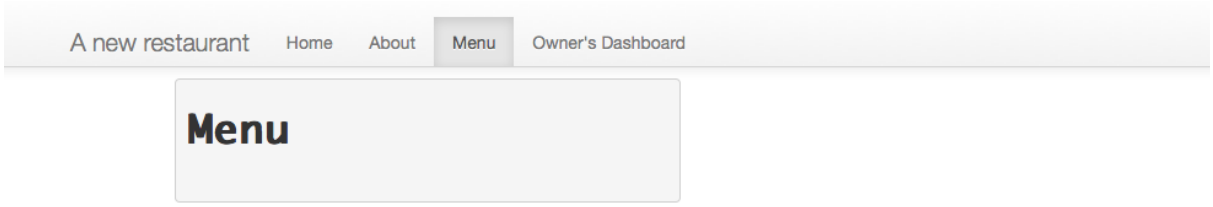
The Main page for GravyXpress is a sleek signup and login page for restaurant owners to create their subdomain within the system. Rendered using a personalized Bootstrap Template, our homepage appears as follows:



A sleek dashboard interface allows the manager to alter the settings of his webpage:

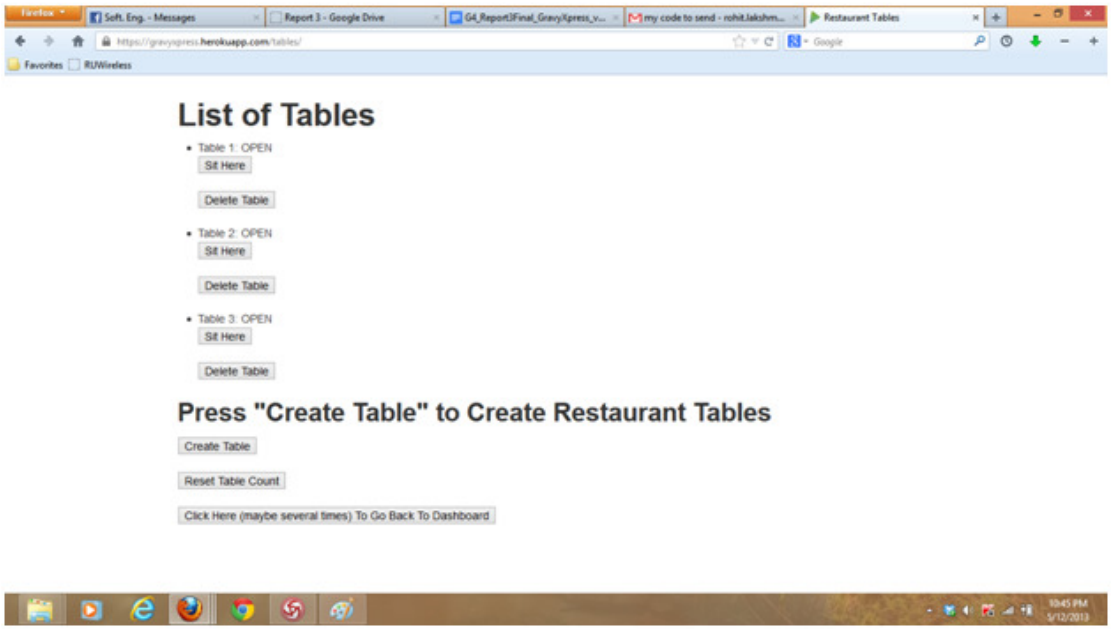


The restaurant itself appears as follows:



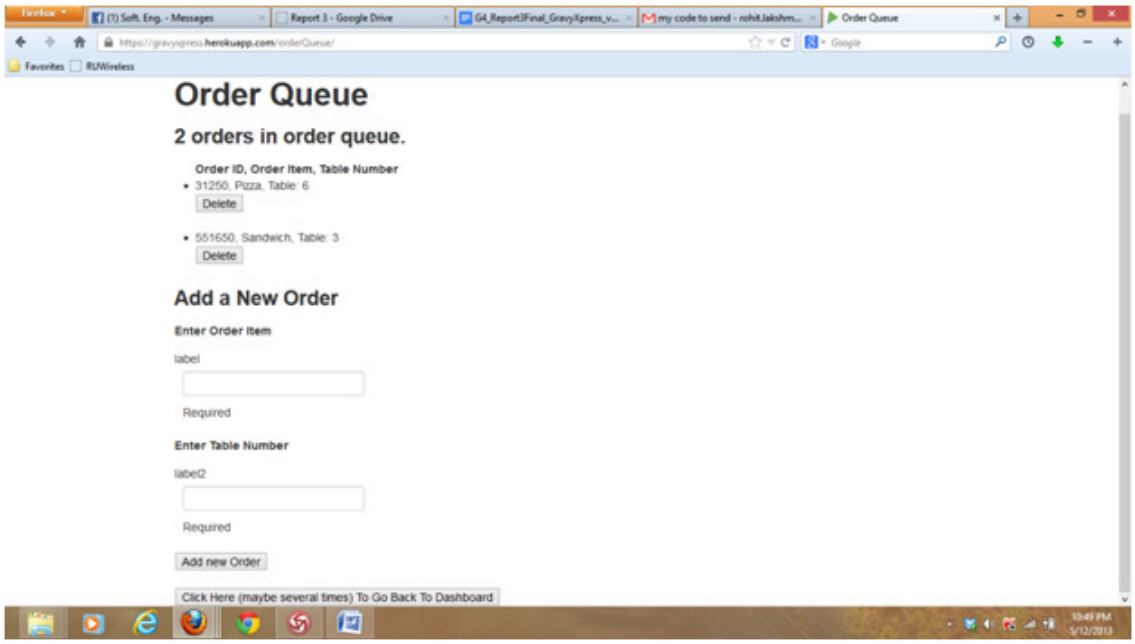
Once again this is a sleek interface that shows the restaurant's personalized webpage.

The following is the Tables Creation and Status page as implemented:

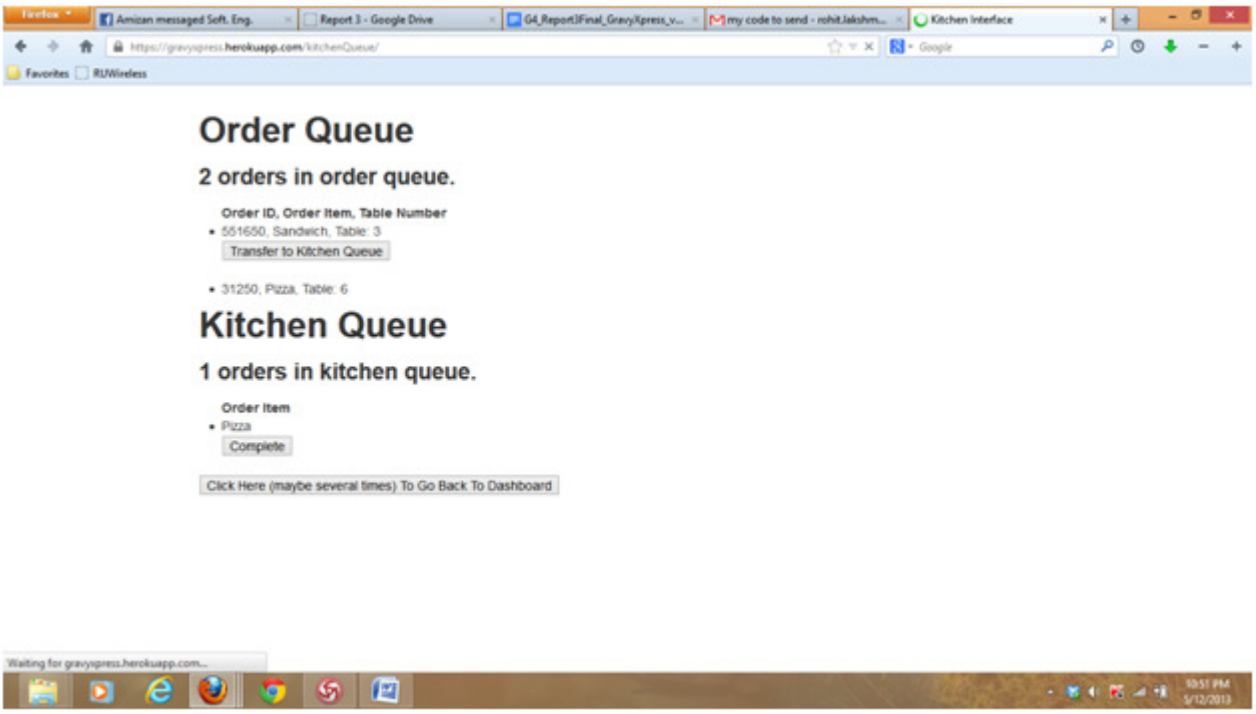


This page will be handled by the Manager who creates the tables and the Waiter who only changes the status of tables. The Waiter will not have access to the bottom 3 buttons.

Next, is the Order Queue where the Customer's orders will be coming in.



These orders will then be pushed into the Kitchen Queue in the Kitchen Interface:



And the waiter will be notified as the order is completed.

On the next page is our envisioned 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 use their own mobile device or a tablet to access the GravyXpress app. Until then, our current application looks sleek. 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

12. 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. Step 2. Enter all information accordingly.	System displays some textboxes that ask for name of restaurant, manager name, hours of operation, and address. 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. Step 2. Enter all information accordingly.	System displays some textboxes that ask for name of employee, type of employee, pay roll for employee, and work schedule of employee 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. Step 2. Select a category to add a new food item. Step 3. Select the "Add New Item" option. Step 4. Enter information and hit the "Add Item" button when finished.	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. 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". System displays pop-up window with some textboxes that ask for the "Name:", "Price:", and "Inventory Count:" of the new item. System adds the item into the list in alphabetical order.

DeleteCategory

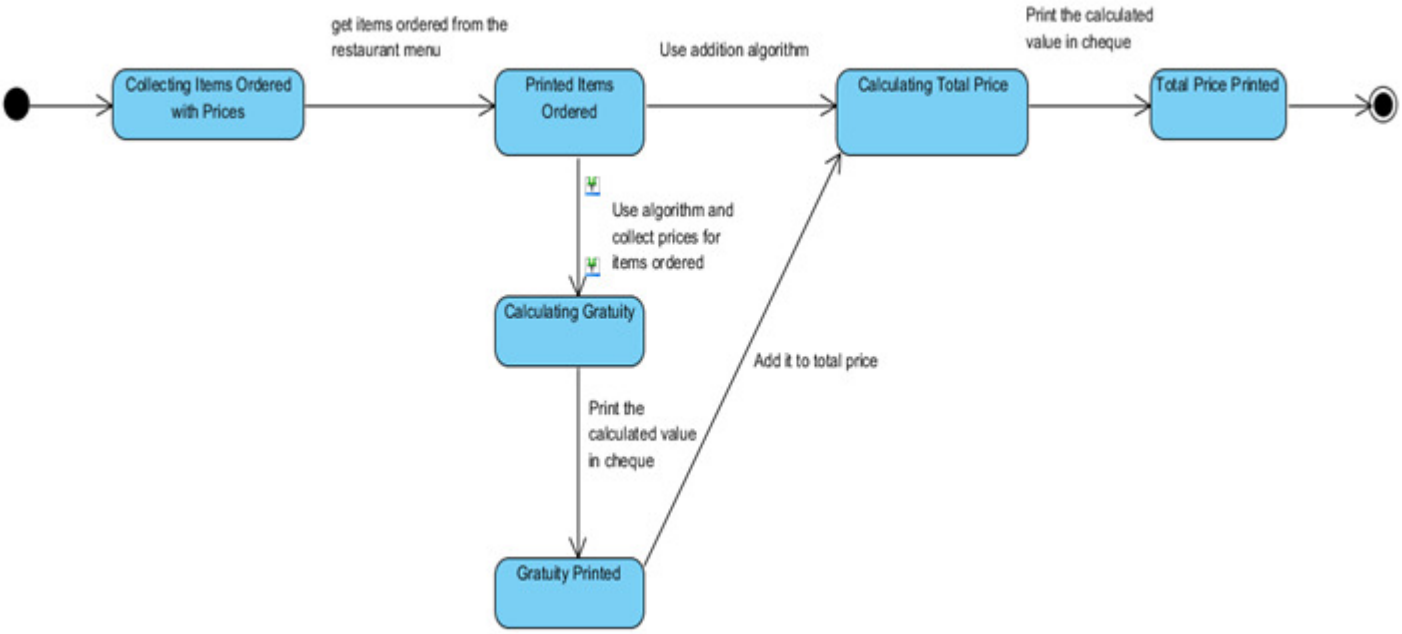
Test-case Identifier:	TC-10
Use Case Tested:	UC-5, alternate scenario
Pass/Fail Criteria:	System passes if the chef is successfully able to delete a category in the restaurant menu.
Input Data:	Delete command.
Test Procedure:	**Expected Result:**
Step 1. Login to chef's interface and select the "Create Restaurant Menu" option. Step 2. Select the "Delete" button next to the category wished to be deleted. Step 3. Select "Yes".	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. 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". System deletes the category and returns to category list.

UpdateMenu

Test Case Identifier: **Use Case Tested:** **Pass/Fail Criteria:** **Input Data:**	TC11 UC-5 alternate scenario System passes if the chef successfully able to update the existing item in the menu Update Command
Test Procedure:	**Expected Result:**
Step1: Logs in as Chef **Step2:** Enter the id number of the current item to be updated w/ updated info **Step3:** Hit "Update"	System displays the current menu to chef System displays the input GUI to update the menu After using the input GUI, system displays the updated menu

B) Unit Tests:

1. Cheque

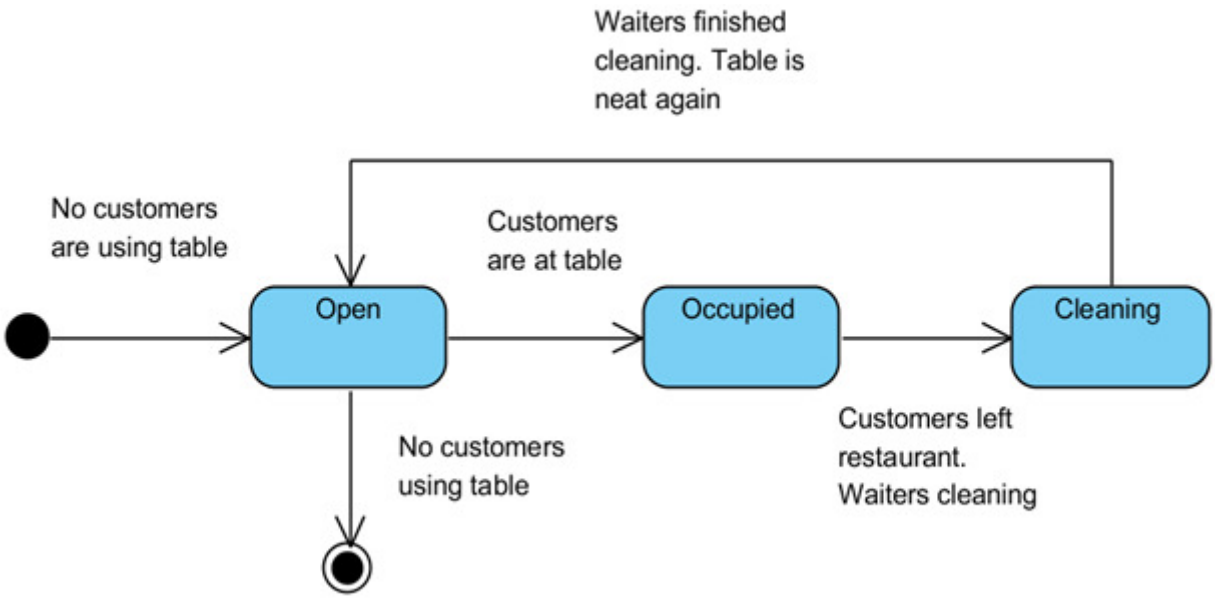


This is the state diagram for the Cheque class. This shows how the cheque for the restaurant customer is calculated and printed on the screen. This includes the total price and gratuity that the customer must pay before leaving. To test all states of this class, we will use the following method calls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

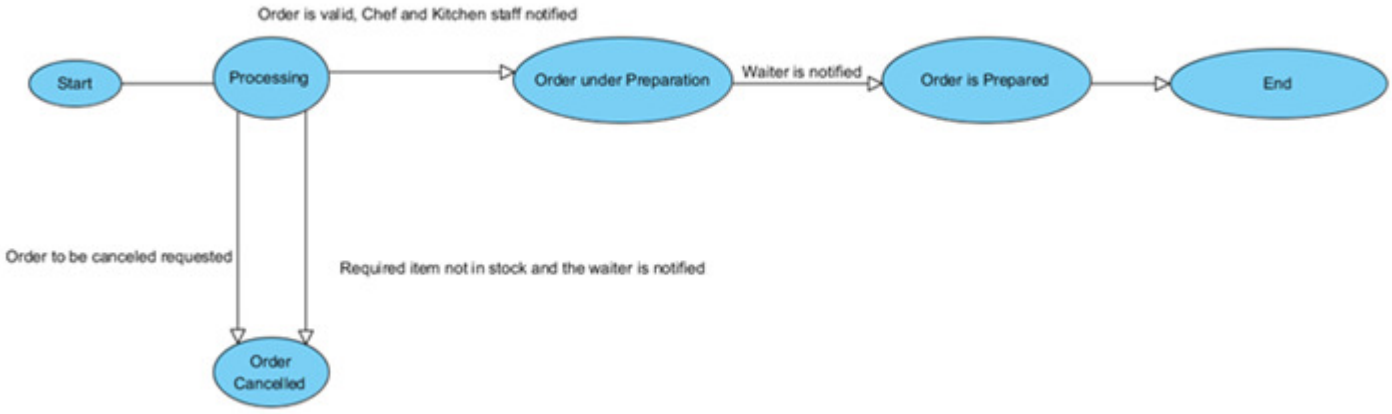


This is the state diagram for the Table class that shows the status of the restaurant table in the system. The following method calls will be used to test every behavior of the Table class:

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.

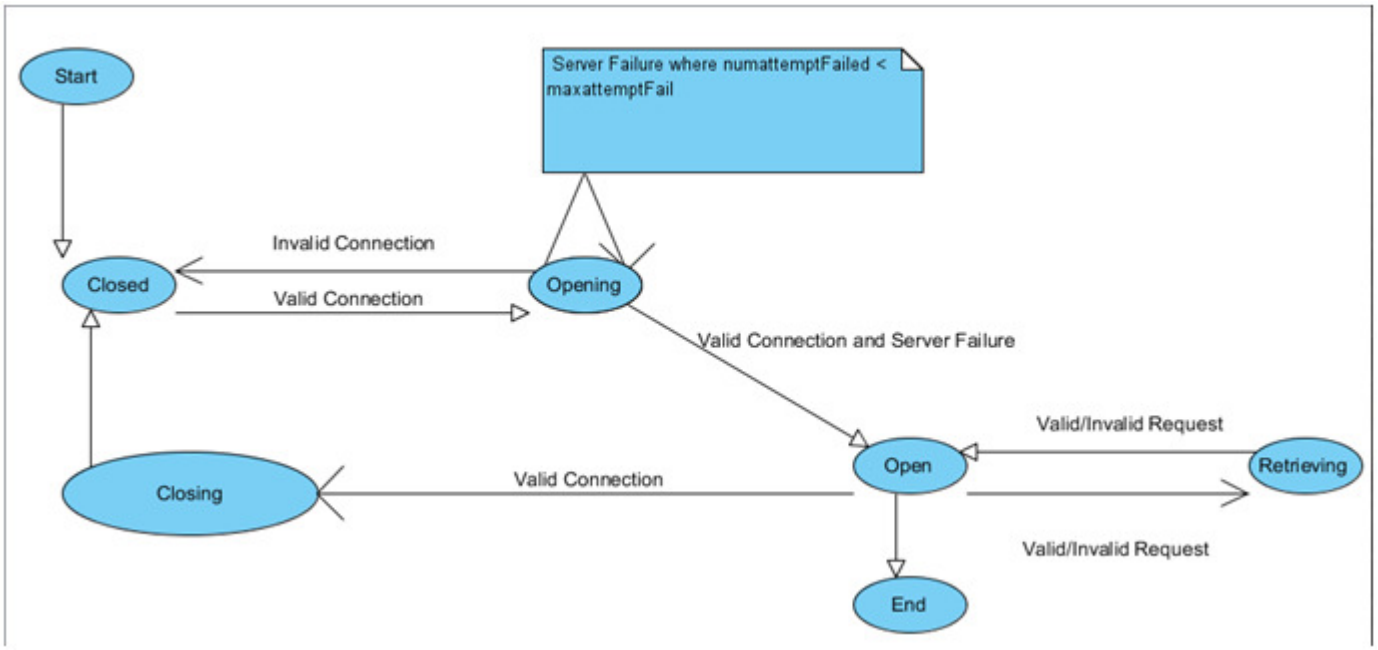
3. OrderFood



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.

13. History of Work, Current Status, and Future Work:

1 For Administrative dashboard, HTTP requests should become asynchronous rather than synchronous. Submitting a form shouldn't clear other forms, and each form should be able to be submitted independently.

2 For now, HTML injection is allowed by the dashboard in the restaurant's about and hours sections. In the future we should probably implement a bbcode style subset of HTML to prevent malicious users from overloading the server. For now, as this is a small website, it is probably okay to leave the users the ability to inject HTML to further customize their own sub-domains and add images and logos to make their web-pages look even better.

3 As described in our first demo, different functions of our project were implemented using different platforms, such as JAVA, MATLAB, ASP.NET and Play! Framework. One should work on these functions that have been implemented outside of the Play! framework since we are using Play! framework as our final intergration platform

4 Currently, the manager is able to create user accounts for his/her employees, and add all relevant information to be stored in the database.. The manager is also able to upload tables with certain seating capacities for his/her specific restaurant that will persist in the database of that restaurant, and can remove them if so desired. In the future, we would like for each employee that logs in to see only the pages pertaining to them (the chef only seeing the order queue, the waiter only seeing the tables he is assigned, etc.)

5 Some members had code implemented in Matlab. Because the goal was to make a webapp using the Play Framework, the code would have been "converted" to Play in the future. Additional use cases that were not implemented due to time constraints would have been done in Matlab and again converted to Play. In addition, more Gantt Charts would have been made and edited to visually show the team deadlines.

6 A more personalized GUI will be implemented down the line. For now we have implemented something sleek, clean and simple.

14. References:

1 General information from:
a. Software Engineering by Ivan Marsic, Rutgers University
http://www.ece.rutgers.edu/~marsic/books/SE/book-SE_marsic.pdf

2 Specific ideas/information:
a. Role system for restaurant employees: borrowed from Group 15 - Spring 2007.
b. Bartender position borrowed from Group 11 - Spring 2012.
c. The floor plan idea, including color coding, from Group 2 - Spring 2012.
d. 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
e. A Youtube video about how to draw sequence diagrams:
http://www.youtube.com/watch?v=18_kVlQMavE
f. A tutorial on using Visual Paradigm for database design:
http://knowhow.visual-paradigm.com/database-design/design-database-with-schema/
g. Microsoft's Website on SQL servers:
http://msdn.microsoft.com/en-us/library/ms143506.aspx
h. Microsoft's Website on screen and resolution settings:
http://windows.microsoft.com/en-us/windows-vista/getting-the-best-display-on-your-monitor
i. YouTube video (and other similar videos) on Gantt Charts in Microsoft Excel:
http://www.youtube.com/watch?v=4Cv_RHWs7cM
http://www.youtube.com/watch?v=sA67g6zaKOE

3 Software used:
a. Google Drive to write the report.
b. Microsoft Visual Paradigm for UML 10.1
http://www.visual-paradigm.com/download/vpuml.jsp?edition=ce
c. Adobe Photoshop CS6 for the UI mockups.
d. Adobe Dreamweaver CS6 for the actual website.
e. Heroku for application deployment to the cloud
f. Java Play Web Framework

4 Pictures:
a. http://www.bubblews.com/assets/images/news/1132949374_1364638619.jpg