Restaurant Automation

Report #3 Part 1

Software Engineering - 14:332:452

Group #14: Jan Matthew Miranda, Kevin Dai, Eric Jiang, Peter Luo, Christian Remolado, Leonardo Roman, Mohammad Sadiq Rehan

Desktop URL:

https://github.com/leonardoARoman/RestaurantAutomationMainSystem

Mobile URL:

https://github.com/janmmiranda/restAuto

Backend Server URL:

https://github.com/janmmiranda/raBackend

*must be running for Mobile Application to work, input following credentials into lines 6-9 in index.js { host: "restauto.c8kfv5fb1sng.us-east-2.rds.amazonaws.com",

user: "restauto",

password: "restauto1",

database: "restauto" }

Summary of Changes

- Summary of Changes Section:
    - Lists all revisions for Report 3
- Section 1: Customer Statement of Requirements
    - Added new problems under Listed Problems (problem 1.5, page 9)
    - Added Project Unique Features (page 10)
- Section 3: System Requirements
    - Added new features to Functional Requirements (page 14)
- Section 4: Functional Requirements Specifications
    - Included new Use Cases (pages 24, 27)
    - We should update our use cases that are going to be implemented in the final demo. So the QR thingy and anything new we want to add. We can also make up use cases for any updates we'd want but don't have time for.
- Section 5: User Interface Specification
    - Updated use case points to change the productivity factor from 24 to 28 per use case point. Resulting in an increase of 1320 hours to 1540 hours required for the system.
- Section 7 Interaction Diagrams: Updated the use cases with Design Patterns.
- Section 13: “Plan of Work” → “History of Work, Current status and Future Work” (page 103)
    - Added completed milestones/deadlines. Compared current milestones to the old milestones.

Effort Breakdown

FINAL BREAK DOWN	Jan	Kevin	Eric	Peter	Christian	Leonardo	Sadiq	TOTAL
1. Customer Statement of Requirements	38%		28%		15%	19%		**100%**
2. Glossary of Terms			60%	40%				**100%**
3. System Requirements	15%	15%		8.33%	8.33%	38.33%	15%	**100%**
4. Functional Requirements	26%	11.66%	6%	10.68 %	10%	26.66%	9%	**100%**
5. Effort Estimation	43.75%		25%	13.75 %	5%	12.5%		**100%**
6. Domain Analysis	50%					50%		**100%**
7. Interaction Diagrams			12.5%	12.5%	20%	25%	30%	**100%**
8. Class Diagrams and Interface Specifications	18.75%		25%	12.5%		18.75	25%	**100%**
9. System Architecture and System Design			33.33%		33.33%		33.33%	**100%**
10. Algorithms and Data Structures	40%			20%		40%		**100%**
11. User Interface Design and Implementation							100%	**100%**
12. Design Tests			60%	20%	20%			**100%**
13. History of Work, Current status, and Future Work	30%	10%	25%			25%	10%	**100%**
14. References			33.33%		66.67%			**100%**
15. Project Management	60%					40%		**100%**
Total (1,500 pts)	**321.5**	**36.7**	**308.2**	**137.8**	**178.3**	**295.2**	**222.3**	**1500**
Individual Percentage	**21.4%**	**2.40%**	**20.5%**	**9.2%**	**11.9%**	**19.7%**	**14.8%**	**99.99%**

Table of Contents

Group Information 1

Summary of Changes 2

Effort Breakdowns 3

Table of Contents 4

1. Customer Statement of Requirements

    Problem Statement 6

    Listed Problems w. Solutions 6

    Proposed Solution 9

    Project Unique Features 10

2. Glossary of Terms 12

3. System Requirements

    Functional Requirements 14

    Nonfunctional Requirements 15

    Ideology of System Architecture 16

    UI Requirements 17

4. Functional Requirements Specification

    Stakeholder 18

    Actors Goals 18

    Use Cases 19

    Traceability Matrix 27

    System Sequence Design 29

5. User Interface Specification

    Preliminary Design 35

    Effort Estimation 48

6. Domain Analysis

    Domain Model 52

System Operation Contracts 58

7. Interaction Diagrams
Use Case Diagrams 61
8. Class Diagram and Interface Specification
Class Diagram 67
Data Types and Operation Signatures 70
Traceability Matrix 80
9. System Architecture and System Design
Architectural Styles 82
Identifying Subsystems 83
Mapping Subsystems to Hardware 84
Persistent Data Storage 84
Network Protocol 84
Hardware Requirements 86
10. Algorithms and Data Structures
Desktop 88
Mobile app 91
Website 93
11. User Interface Design and Implementation 94
12. Design of Tests 96
13. History of Work, Current status and Future Work 103
16. References 106

Customer Statement of Requirements

Problem Statement

As more and more people decide to eat out instead of cooking at home, restaurants are continuing to grow in size and numbers. As they continue to expand, their vulnerabilities become more and more exposed. Restaurants have been around since the 11th century some historians say, and although foods, equipment and aesthetics have changed, the process of running a restaurant hasn't. As restaurants continue to gain more patrons, some are looking at expanding staff and floor space to house more patrons and serve them. However this solution doesn't truly solve the issue, it's just multiplying the inefficiencies they had since they opened. However we believe our Restaurant Automation Software is the solution for expanding restaurants how want to truly become efficient in the food catering business. We aim to solve problems restaurants have such as table management, communications between table orders and the kitchen, easy archiving of sales, workers recorded hours worked and more through technology. We believe restaurants who are expanding and plan on taking a larger amount of patrons should adopt our Restaurant Automation Software to increase their efficiency, save money and time.

Listed Problems

1.1 <u>Difficulty with Accessing Restaurant Information</u>

Problem:

Many customers come to a restaurant wondering about many minute details that take up time to ask hosts and hostesses upon their visits. Although visitors' questions may be small and quick to answer, being constantly asked the same questions can lead to lost time that could have been used for other host tasks such as checking reservation statuses and checking up on table statuses. Because time is money is the big restaurant industry, this ultimately leads to inefficiency.

Solution:

The website displays very transparently the menu, contact information, current offers/discounts. This way, the website provides a map of tables with availability and a login account with user information to check-in conveniently online.

1.2 Maintaining Table Availability Status

Problem:

When customers first enter a restaurant, they are usually greeted and asked how many people are within a customer's party so that they can be put in a paper-to-pen queue for a table. After that, hosts have to keep an eye out for when a table opens up physically around the restaurant in order to update a whiteboard, diagrammed layout of the restaurant which shows all tables and statuses. This is problematic as hosts spend a lot of time trying to update and transcribe the queue list and the whiteboard, which can lead to errors. On top of that, the hosts don't have a lot of time greeting the guests to make them feel welcome to the establishment.

Solution:

To reduce the amount of mental and writing work for the hosts/hostesses, a computerized algorithm will be introduced to find available tables and seat guests accordingly. The computer will ask for two simple inputs: the customer's name and how many people in the customer's party. Then the algorithm will place parties in separate queues based on how big the party is (how many people are in the party). Then, the queues will be prioritized by who arrived and got the host/hostess to input their party information first. Then, when a table is ready, a notification will come up on the host or hostess' device, displaying which table is ready for which customer party in the appropriate queue. At that point, the host or hostess can confirm on the device if they will seat the customer's party at the recommended table.

This also beneficially allocates social and emotional interactions for hosts and hostesses to have with their customers to make the latter feel welcome, rather than potentially stressing hosts and hostesses with laborious writing and constant organizing.

1.3 Customers Waiting to Pay the Check

Problem:

Most of the time, customers are looking to pay the bill immediately after they finish their meal. However, waiters often prioritize taking orders and serving dishes, putting the check in the backburner of their minds. Furthermore, for the check to be finalized, it has to go through a gruesomely long process. First, the check has to be prepared by the waiter, then brought to the customer, stalled by the customer's decision on how to pay, then stalled again until the waiter comes back to take the check back to the register This process can be even more prolonged if the waiter is waiting tables during high-traffic hours, making the transporting times for the check even longer.

Solution:

Instead of taking the orthodox route of transporting checks, a tablet-based payment system can be introduced to cut down on the time customers have to wait for the check. In this system, a tablet is introduced at each table so that when a party is all finished, the customers can then choose when and how to pay. The tablet would then be able to access and display the table's ordered items with their individual prices, a subtotal, the total tax, and an option to tip the waiter electronically. Finally, the electronic payment system will ask for cash, credit, or debit payments and respond accordingly with correct change and a physical receipt.

1.4 Attendance Punctuality

Problem:

Obsolete clock in/out methods such old punch cards, or ID number typing, or even swiping badges can not keep employees from cheating the time they clocked in/out. There are cases in which employees who are running late ask their fellow/friend employee to clock them in so there can not be any record that such employee was late for work.

Solution:

Fingerprint recognition would solve this integrity problem. Every employee must use their fingerprint to clock in/out as part of self identification.

1.5 Uncertainty of Daily Operations.

problem:

There are occasions in which manager has to guess, by experience or inspection, which days might be more busy. Typically weekends are the target key in order to make decisions, such as how many workers might need to be schedule for work. But the truth is, there is no way of knowing this with 100% certainty. The probabilities that a regular day might be a crowed one, whether is a weekend/holiday or not, exists.

Solution:

To build a predicting model, that will learn from previous restaurant data, in order to have a probabilistic estimation about the future. Predictions may include, crowded days, popular dishes, customers similarities, and even scheduling. This solution will be developed using python and some modules such as pandas, sklearn and more.

Proposed Solution

With our solution, we aim to address the problem of restaurants being inefficient and held down by the non-optimized systems workers use to operate the restaurant. One of the biggest problems we wanted to address was the wait time between a customer's meal and the time they pay the check. Our system seeks to improve the traditional system through automating the systems through electronic means; the customer will be seated based off an electronically maintained table status window, the customer's orders will be sent electronically to the kitchen, meal status will be electronically notified to waiters and payment can be handled electronically by the table if through credit/debit.

Our proposed automated system will be available to restaurant owners through a software suite that contains a main desktop application, a website and a mobile application. Each application will be responsible for different systems running in the restaurant. The main application will be charge of maintaining table status, assigning waiters to tables, viewing archived orders, and clocking in workers' hours. The mobile application will handle sending of the meal orders to the kitchen via electronic notifications, notifying waiters of meals ready to serve, and paying checks through credit cards. The website will be a portal to captivate potential

customers through previewing the menu, displaying photos of the food, and providing a simple reservation system.

Through this software suite, restaurant owners will be able to increase overall efficiency of the restaurant from all systems. Not only will it increase the speed at which a customer can sit down, eat, and pay but it will also increase the efficiency of managers and accountants. The archival capabilities of clocking in workers' hours and recording sold meals will allow for easy payment to employees as well as accurate information of revenue gained, respectively.

A restaurant's main purpose is to give customers a place to sit and eat delicious meal giving the customers a sense of convenience of just having to eat and pay. Our app will increase efficiency in and outside of the kitchen, which in turn will increase customer satisfaction, reduce workers responsibilities, and generate the restaurant more revenue.

Project Unique Features.

What makes our product different from existing projects ? Previously in this course, restaurant automation proposed solutions for a software application have been multithreaded, or used some sort of network communication (TCP/IP) or even used a cloud server. While other projects have focus in one software model say, website, mobile application or a desktop app. Our group intents to use all three approaches to divide the system and conquer different type of audience. Moreover, what really separates our product from others, is our intention to combat any attendance deficiency. Our system proposes a solution to verify each employee clocks in and out of their shift of duty by using biometric recognition using their fingerprint.

We all know how learning from the past is a big thing nowadays. Our project will intend to use all "generated" data to make a prediction about the future. The idea is to use restaurant history in order to predict if the next day, week or holyday will be a busy/ier or slower day. The predicting model, using some machine learning tools, will help restaurant staff members to better prepare themselves for a crowded/slower day. This means, having a visual estimation on whether to schedule more staff to work on a particular day, order more condiments for a particular dish according to its popularity or even the opposite. The plan also involves searching for relevant

data in order to train our predictive model. But the idea is to use the actual restaurant data, generated from the past, to learn from it and to provide support on decision making.

In addition to speeding up systems for service time our system seeks to speed up the payment process for restaurant customers who wish to pay digitally. Our mobile application utilizes Braintree Payments, a technology for online payments that other groups in the past have not used (past groups used proprietary technology from Paypal themselves). Braintree Payments allows waiters/waitresses to accept payment from customers via Credit Card, Paypal and now a new payment service Venmo, which no group has implemented before. By allowing restaurant goers to pay with new yet widely used digital payment processes, restaurants who use our software can get paid faster and give restaurant goers a faster experience.

Glossary of Terms

Technical Terms	
Database	The file where the menu items, inventory, scheduling and orders are stored.
Employee Portal	Schedule accessible by employees that lists their respective shifts and any open shifts they may cover.
Restaurant Automation	Use of an internal restaurant management application to automate and carry out major operations within a restaurant establishment
Graphical User Interface (GUI)	A type of user interface that allows users to interact with electronic devices through graphical icons and visual indicator
Table Availability Schedule	A database that shows the availability of the tables in the restaurant
Order Progress Queue	A priority queue that shows the progress of each order and what will be prepared first.

Non-Technical Terms	
Foodies	A person who has an ardent or refined interest in food and alcoholic beverages. Seeks food experiences as a hobby.
Bartender	Serves beverages and maintains the supply and inventory of the bar
Bill	A statement which contains details of the menu items that have been purchased and the money that is owed to them
Busboys	Clears tables, takes dirty dishes to the dishwasher, and sets the tables
Chef	Responsible for creating and planning menus, overseeing food preparation, and supervising the kitchen staff
Chef Hotline	The functionality that allows customers to contact the chef from the waiter's tablet to inquire about menu items

Customer	Any person that orders an item from the menu or walks into the restaurant. He/She can order food and place reservations online.
Customer Satisfaction	Measurement of how food preparation, customer service, and overall experience meets or surpasses customer expectation
Dine-in	When a customer would like to be seated and eat in the store
Host/Hostess	The person to greet and seat a party
Manager	The person that is responsible for inventory management, employee scheduling, payroll and customer satisfaction
Menu	A list of food and beverage offered to the customer
Profit	Revenue subtracted by expenses
Party	The customer and their guests, if they have brought any. A party is sat together.
Reservation	An arrangement made ahead of time for the party to dine at the restaurant
Take-out	Orders that are prepared for customers to be eaten outside of the restaurant
Waiter/ Waitress	Employee that takes customers' orders, brings completed orders to customers, and marks recently vacated tables

System Requirements

Functional Requirements

Identifier	User Story	Story Point
REQ-1	As a hostess, I can search and update available tables quickly	4
REQ-2	As a hostess, I can update reservations if requested by guest	2
REQ-3	As a waiter/waitress, I can quickly send table orders to the kitchen without having to go their myself	6
REQ-4	As a waiter/waitress, I can promptly be notified of when meals are ready to be sent from the kitchen to the table	4
REQ-5	As a waiter/waitress, I can easily let the hostess know of when tables are available for new customers without going to the hostess directly	3
REQ-6	As a waiter/waitress, I can keep track of which meals are to be served at the respective tables	2
REQ-7	As a waiter/waitress, the bill can be provided quickly and accurately as the order history and cost will be stored on the system without errors.	4
REQ-8	As a cook, I can see in queue orders and prepare them accordingly	4
REQ-9	As an employee, I can clock in and out of work	4
REQ-10	As a manager, I can make changes on orders if needed	3
REQ-11	As a manager, I can post staff work schedules	4
REQ-12	As a manager, I can see all transaction reports such as daily sales, profits, payments etc	4
REQ-13	As a manager, I can see how long each waiter worked by the amount of time they were logged into their tablet	3
REQ-14	As a customer, I can make reservations	3
REQ-15	As a customer, I can find information about the restaurant such as menu, hours of operations, etc	2

REQ-16	**As manager, I can make an estimated guess about how many workers I will schedule to work**	2
REQ-17	**As chef, I can make an estimated guess about what condiments I will use more, what dishes are more popular.**	2
REQ-18	**As a waiter/waitress, I can accept payments from customers through credit cards, Paypal or Venmo, from the customers table**	3

Nonfunctional Requirement

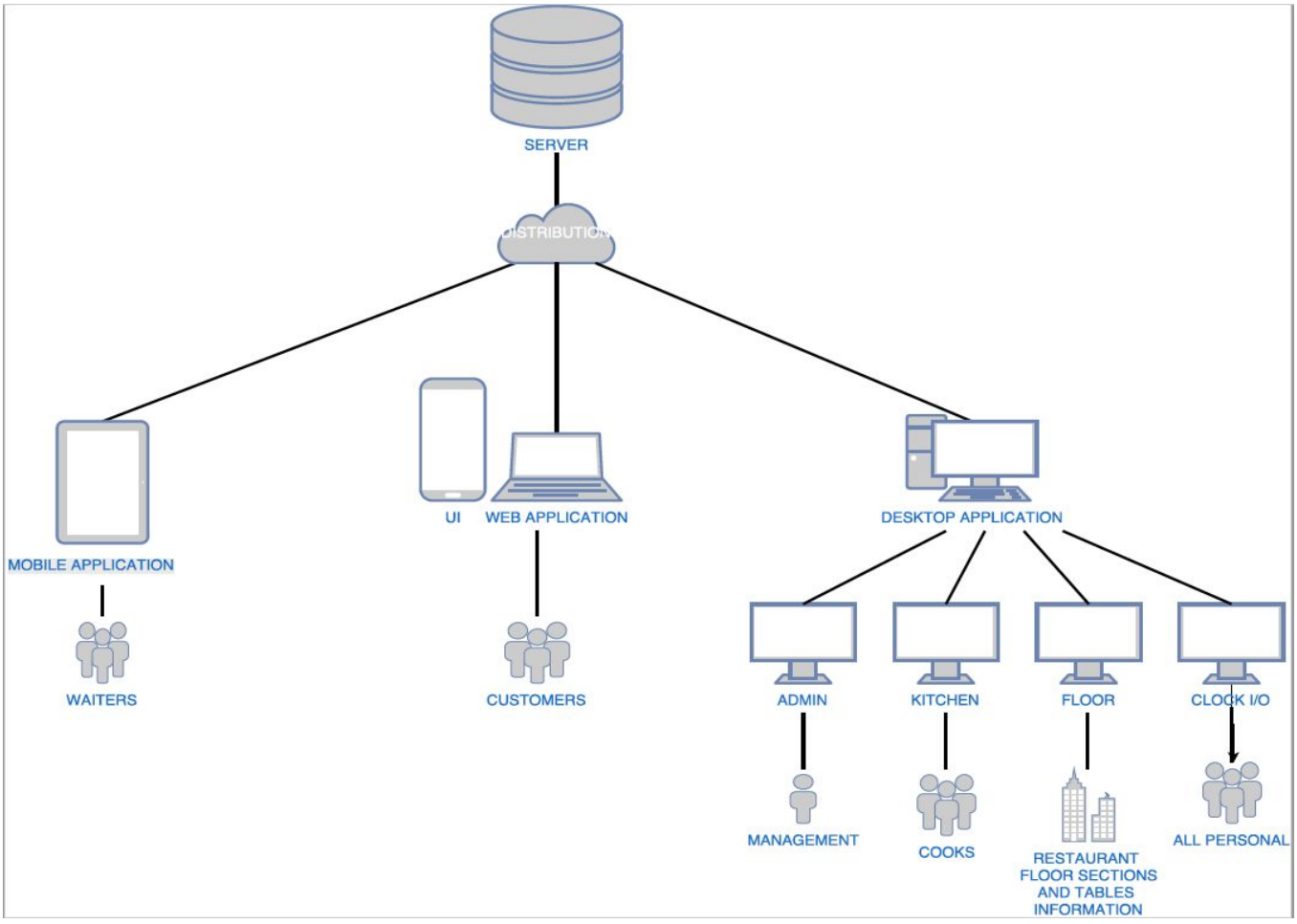


Figure 1.1 The System Architecture of the Restaurant Software Suite

FURPS	
Functionality	To divide the system-to-be into subsystems independently of each other in such way that each subsystem will meet all requirement satisfaction.
Usability	For targeted users such as customers, staff and management.
Reliability	For all users with their respective request such as making reservations, staff scheduling, table designation etc.
Performance	Input/Output transactions such as payments, table state, orders, queries, etc.
Supportability	Client/Server interprocess communication via cloud/server

Ideology of System Architecture

Before designing the system-to-be, architectural decision was made, such as dividing and conquer. The main system was divided into three subsystems to target all subdomains(subproblems) and solve them, accordingly to all requirements previously gathered, in order to deliver the final product. Furthermore, desktop application was divided even further in order to solve different challenges in the restaurant domain. As we can see in picture 1.1, desktop subsystems are kitchen, floor, administration and clock in/out devices and each subsystem have their respective architecture.

- Model View Controller(MVC) architecture for mobile devices such as waiter's tablets and desktop app.
- Client/Server architecture for web app and kitchen/waiter intercommunication for meal orders.
- Central repository database architecture(Admin) for keeping a log of data such as clocking in/out times, daily sells, and other essential data.

Although every system device shares same server in other to maintain communication, each subsystem is independent of other subsystems. Different architectural decision was made to target different aspects of challenges a restaurant might face.

UI Requirements

UI REQ-1	The first screen of the system will be the login screen.
UI REQ-2	The tablet interface will be used by waiters/waitresses and will be focused on taking orders.
UI REQ-3	The tablet has a table button to show the table layout of the restaurant. User can see table availability and be able to select the tables that are his/her responsibility.
UI REQ-4	Once a table is selected, a menu will open up where menu items that the customer requests can be selected to add to the customer order.
UI REQ-5	Once the order is final, it can be sent to the kitchen by selecting the place order button. When placing the order, the cost of the meal is also displayed and stored.
UI REQ-6	The tablet has a notification system to notify users when meals are ready to be sent from the kitchen to table.
UI REQ-7	The tablet has a notification system to notify users when tables need to be cleaned.
UI REQ-8	The tablet has a login system to associate each tablet with their respective user.
UI REQ-9	If a manager logs in, the screen presented to him/her will have more functionality such as options to view transaction reports, employee information as well as change employees schedules and make changes to the menu
UI REQ-10	The interface on the kitchen side will allow cooks to view orders and an option to notify the waiter/waitress when a dish is ready.
UI REQ-11	The website will allow customers to view the available reservations calendar and subsequently to create a reservation with a phone verification step included.
UI REQ-12	The website will have basic restaurant information such as the phone number, location, and hours of operation
UI REQ-13	The website will have an interactive menu where pictures of each item are included
UI REQ-14	Mobile Application will have a button that leads to Braintree’s Payment processing portal

Functional Requirements Specification

Stakeholder

Several types of stakeholders exists in our system from restaurants who wish to adopt our system to the actual workers who will follow the new system in it. The system will help speed up efficiency with organization utility for the staff. Those who will specifically be working with the system would be hosts, managers, waiters, kitchen staff and busboy. In addition, customers However customers would be another stakeholder who does not directly make use of the system other than through the website.

Actors and Goals

Initiating Actors:

Busboy		Chef	
Role	Employee who keeps cleanliness of the restaurant	Role	Employee who prepares meals and maintains the kitchen
Goal	To help maintain table status and clean tables marked as dirty	Goal	Manage Inventory of ingredients, manage menu and prepare ordered meals
Customer		Manager	
Role	A patron who orders a meal at the restaurant	Role	Employee who manages the other employees at the restaurant
Goal	To order, eat and pay for a meal	Goal	Manage scheduling, tracking of profits/losses and inventory management
Host		Waiter	
Role	Employee who greets and sits down customers	Role	Employee who interacts with and serves customers
Goal	Seats customers to an appropriate table and assigns waiters to table	Goal	To take orders from table to kitchen and serve the food back to table, also handles customer’s payment

Participating Actors:

Database

Role: To store all information needed for the restaurant system.

Goal: Store, modify and query needed data for Actors to perform their roles.

Employee Tablets

Role: To enable interoperability between the database and waiters.

Goal: Retrieve data from appropriate Actors during their roles.

Main Desktop

Role: Central system to interact with database, tablets and website.

Goal: To have access and control of main systems of restaurant automations, to be controlled by admins (managers).

Website

Role: Provide information about the restaurant to Customers

Goal: Make it easier for customers to find the menu, hours and open reservations.

Use Cases (Casual Description)

Use Case	Name	Description
UC-1	Order	Waitress can create an order of menu items for customers and send it to the kitchen via mobile application
UC-2	Serve	Cooks can electronically notify waitress of when their meal orders are ready to be served
UC-3	Seat	Host/waitress can queue up patrons and seat them to a table using an electronic diagram of the tables to quicken the process
UC-4	Reservation	Customers can use the website to setup reservations for when they visit the restaurant. Hosts can then confirm upon visit
UC-5	Clock in/out	Any Restaurant staff member can log their attendance of work
UC-6	Admin	Managers can have full access to restaurant software functionality(change menu items, check reservations, check worker attendance, check receipts, **make predictions**, etc)
UC-7	Payment Process	Waiters/waitresses can accept payments from customers using their mobile tablet through CC, Paypal or Venmo

Use Cases Full Descriptions

UC-1: Order

Related Requirements:	REQ3, REQ8, REQ10
Initiating Actors:	Any of: Waitress, Cook, Manager
Actor's Goal:	To place the order of the customer as fast as possible, and make sure the cook gets the order correctly
Participating Actors:	Database, Employee Tablets
Preconditions:	*The set of valid keys (orders) in the system is initially 0 *The system queues each of the orders in chronological order and lists the orders made in groups as an array
Postconditions:	*The system starts dequeuing the orders as the cook serves them out and the waitress approves of the order
Flow of Events for Main Success Scenario:	1. Customer enters the restaurant and selects from the menu via an 'order' option 2. System via waitress a) signals to cook to begin order b)signals to manager to oversee order 3. Cook creates order and signals to waitress to retrieve order

Extracting the Attributes

Concept	**Attributes**	**Attribute Description**
Communication	Order request	Used to send orders to the kitchen.
Communication	Order display	Used to display order list on the screen.
Notifier	Tracking order	To display list after updating orders.
Archiver	Tracking number	To stored record of orders in database.

Use Case UC-2: Serve

Related Requirements:	REQ4, REQ6, REQ7
Initiating Actors:	Any of: Waitress, Cook
Actor's Goal:	To serve the order from the kitchen to the respective customer's table as fast as possible
Participating Actors:	Database, Employee tablets
Preconditions:	*The set of valid keys (orders) in the system is NOT zero *The set of valid keys are dequeued in chronological order to be served to the respective customers
Postconditions:	*The set of valid keys at the end of the day is zero (all orders have been completed)
Flow of Events for Main Success Scenario:	1. Cook receives order from waitress and produces meal 2. System notifies a)waitress that the meal is prepared and ready to serve 3. Waitress serves meal to respective customer's table 4. System signals to waitress when a customer has completed their meal 5. Waitress serves bill to respective customer's table

Extracting the Attributes

Concept	**Attributes**	**Attribute Description**
Communication	Order state	To notify waiter when order is ready.
Notifier	Tracking order	Used to show which order is ready to go.
Archiver	Tracking number	To add order in database after is being successfully delivered.

Use Case UC-3: Seat

Related Requirements:	REQ1, REQ5
Initiating Actors:	Any of: Hosts, Waitress
Actor's Goal:	To seat the waiting customers as fast as possible
Participating Actors:	Database, Employee tablets
Preconditions:	*The set of valid keys (tables) in the system is initially 0 *The system queues each of the customers in chronological order and lists the orders made in groups as an array
Postconditions:	*The system starts dequeuing the orders as the waitress begins seating them
Flow of Events for Main Success Scenario:	1. Customer enters the restaurant and selects the 'seating' option 2. Host enters customer party onto the database queue 3. Database calculates the party size with the seating options to optimize availability. 4. The system notifies the host that a table is available so they can direct the party to the table. 5. After seating the party, the host confirms to the system that the party has been seated. 6. The database then moves the party to seated party list

Extracting the Attributes

Concept	**Attributes**	**Attribute Description**
Search request	Table availability	To find which tables are available.
Investigating request	Records list	To query database to process search request.
Communication	Table state	Used for communication between hostess and waiters.

Use Case UC-4: Reservation

Related Requirements:	REQ 2, REQ 14, REQ 15
Initiating Actors:	Any of: Waiters, Managers, Hosts, Customers
Actor's Goal:	To create and manage restaurant reservations
Participating Actors:	Database, Employee tablets, Website
Preconditions:	*The desired table is not reserved by another person for the desired time. *The restaurant is open at the desired time *The Actor must have permission to manage a reservation after it has been created. Therefore, a customer can not delete a different customer's reservation but a manager could.
Postconditions:	*The reservation doesn't interfere with any other reservation and prevents other reservations from being made at this table and time. *Reservation has verified contact information attached to it
Flow of Events for Main Success Scenario:	1. Customer goes on website to the reservations page 2. Customer finds open reservation 3. Customer enters email or phone to verify their identity

Extracting the Attributes

Concept	**Attributes**	**Attribute Description**
Archiver	Update reservation	Used to make any changes on a reservation.
Archiver	Make reservation	Prompts the user to make a reservation.
Search request	Records list	Used to display restaurant information.

Use Case UC-5: Clock in/out

Related Requirements:	REQ 9
Initiating Actors:	Any of: restaurant staff
Actor's Goal:	To keep a log of employees attendance.
Participating Actors:	Database system: To store and keep records of each staff member past working hours.
Preconditions:	*Fingerprint recognition constraint in order to clock in and out by staff member.
Postconditions:	*Clocked in/out time log and storage.
Flow of Events for Main Success Scenario:	1. Employee hits clock in/out button. 2. Employee places finger for fingerprint recognition. 3. Clock in/out granted. 4. Time is recorded and given to respective staff member

Extracting the Attributes

Concept	**Attributes**	**Attribute Description**
Search request	User's identity	Prompts user to enter credentials.
Investigating request	Records list	Used to query database for credentials matching.
Archiver	Log time(in/out)	Used to send information to database server in order to be saved.
Notifier	Display time(in/out)	Displays user's time in or out on the screen.

Use Case UC-6: Admin

Related Requirements:	REQ 10, REQ 11, REQ 12, REQ 13, **REQ-16, REQ-17**
Initiating Actors:	Any of: Management personal(Chef, head manager, supervisors)
Actor's Goal:	To make management **decisions, predictions** and changes such as menu, staff schedules, payments, orders.
Participating Actors:	Database, Main Desktop
Preconditions:	*Login as administrator. *Username and password.
Postconditions:	*Save any changes made to database system.
Flow of Events for Main Success Scenario:	1. Manager enters to administration system. 2. Manager enters username and password. 3. Changes may be made. 4. Changes may be saved. **5. Data analysis may be visualized.**

Extracting the Attributes

Concept	**Attributes**	**Attribute Description**
Archiver	Set orders	To fix any discrepancies in a particular order.
Archiver	Set schedules	Used to upload staff working schedules.
Search request	Get data	To get lists of interesting records selected for further investigation.
Notifier	Table status	Used to estimate how long a table has been occupied.
Notifier	Daily operations	Used to estimate how busy a day might be.

Use Case UC-7: Payment Process

Related Requirements:	REQ 13
Initiating Actors:	Waiter/Waitress
Actor's Goal:	To make payments for meals through Braintree's Payment process
Participating Actors:	Database, Employee's Tablet, Braintree Server
Preconditions:	*Waiter/waitress is logged into employee tablet *Customer is done making orders and ready to pay
Postconditions:	*Accept payment from customer
Flow of Events for Main Success Scenario:	1. Waiter/waitress clicks on Pay button 2. Customer chooses payment process 3. Customer proceeds to fill out payment process 4. Upon successful transaction, restaurant can see transaction history on Braintree site

Extracting the Attributes

Concept	**Attributes**	**Attribute Description**
Archiver	Pay request	Open payment process UI
Archiver	Set payment	Set payment process method
Send request	Send Transaction Data	Process payment
Notifier	Get Transaction Data	Receive payment process result

Traceability Matrix

System Requirements to Use Cases

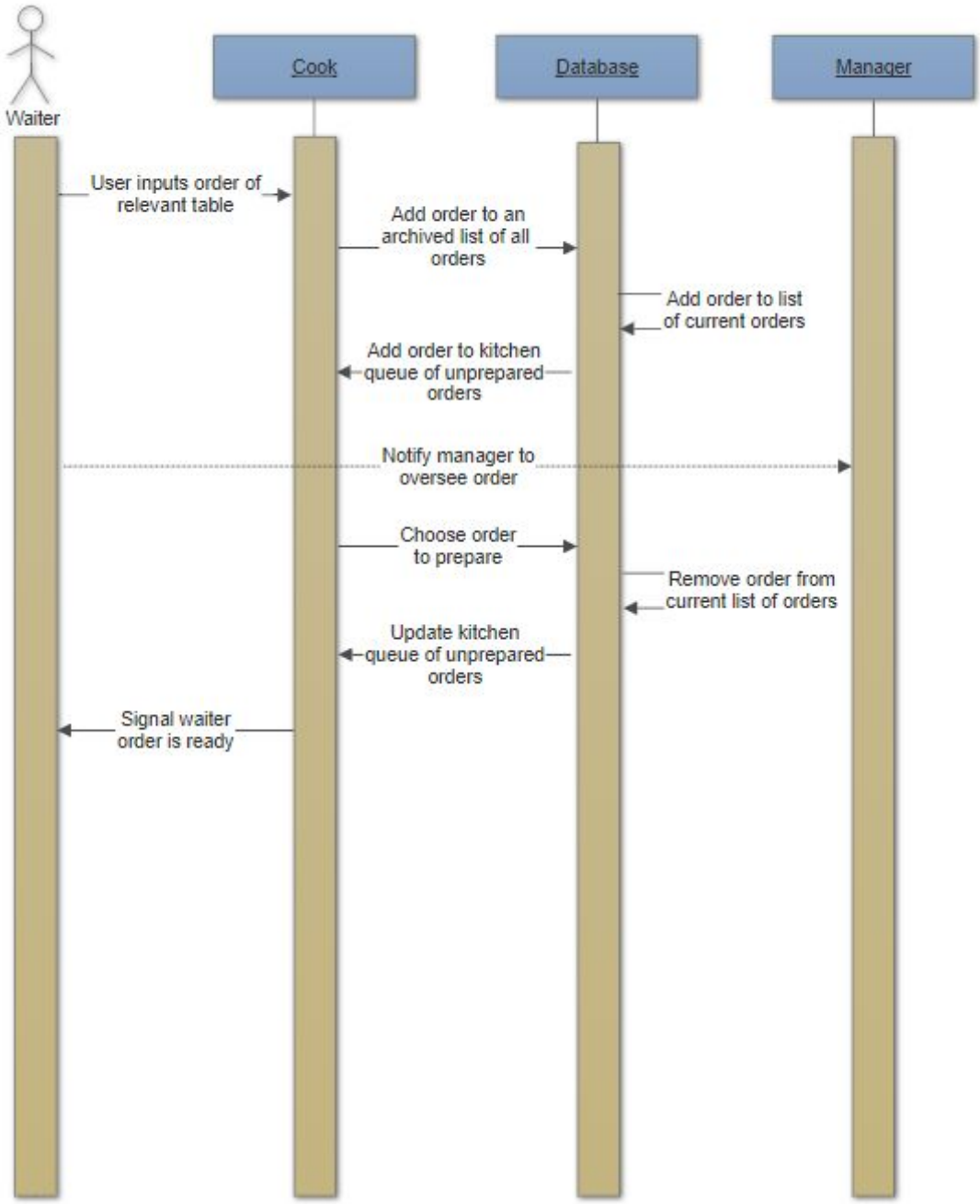
Requirements	**PW**	**UC1**	**UC2**	**UC3**	**UC4**	**UC5**	**UC6**
REQ1	4			*			
REQ2	2				*		
REQ3	6	*					
REQ4	4		*				
REQ5	3			*			
REQ6	2		*				
REQ7	4		*				
REQ8	4	*					
REQ9	4					*	
REQ10	3	*					*
REQ11	4						*
REQ12	4						*
REQ13	3						*
REQ14	3				*		
REQ15	2				*		
REQ16	2						*
REQ17	2						*
Max PW		**6**	**4**	**4**	**3**	**4**	**4**
Total PW		**13**	**10**	**7**	**7**	**4**	**18**

Use Cases to Domain Model

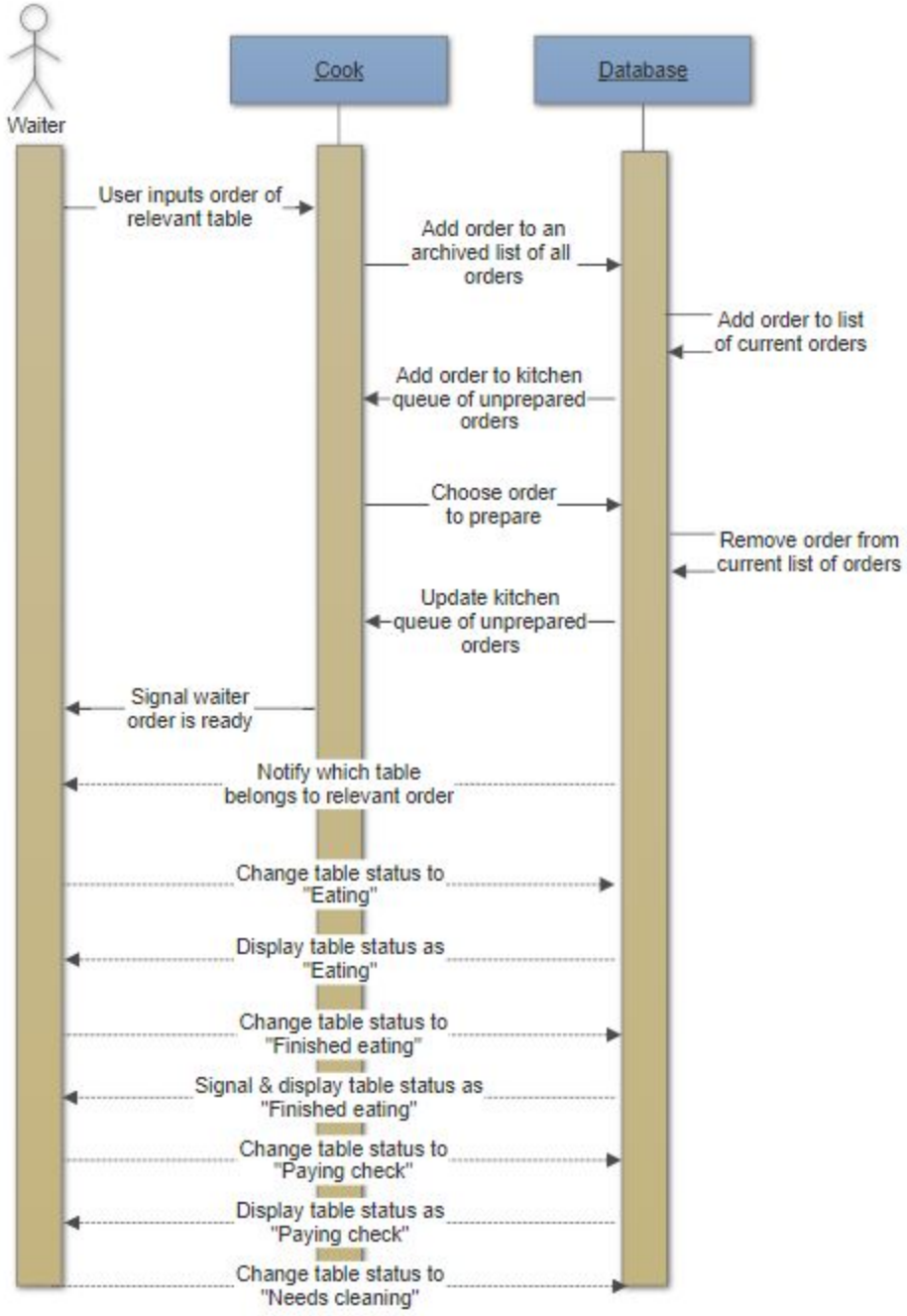
Use Case	**PW**	Communication	Search request	Investigating request	Archiver	Notifier
UC1	13	*			*	*
UC2	10	*			*	*
UC3	7	*	*			
UC4	7		*		*	
UC5	4		*	*	*	*
UC6	18		*		*	*

System Sequence Diagram

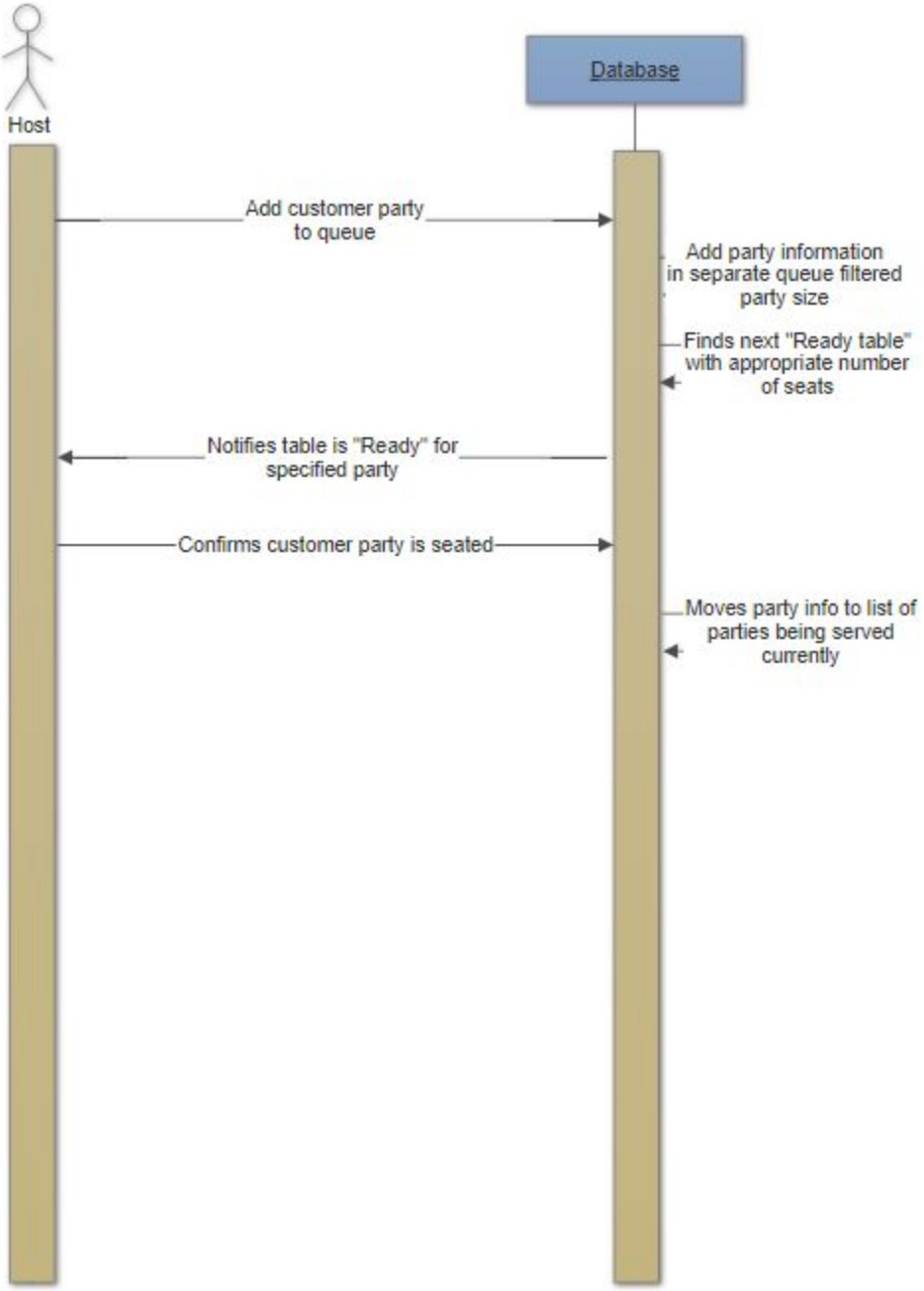
Use Case 1: Order



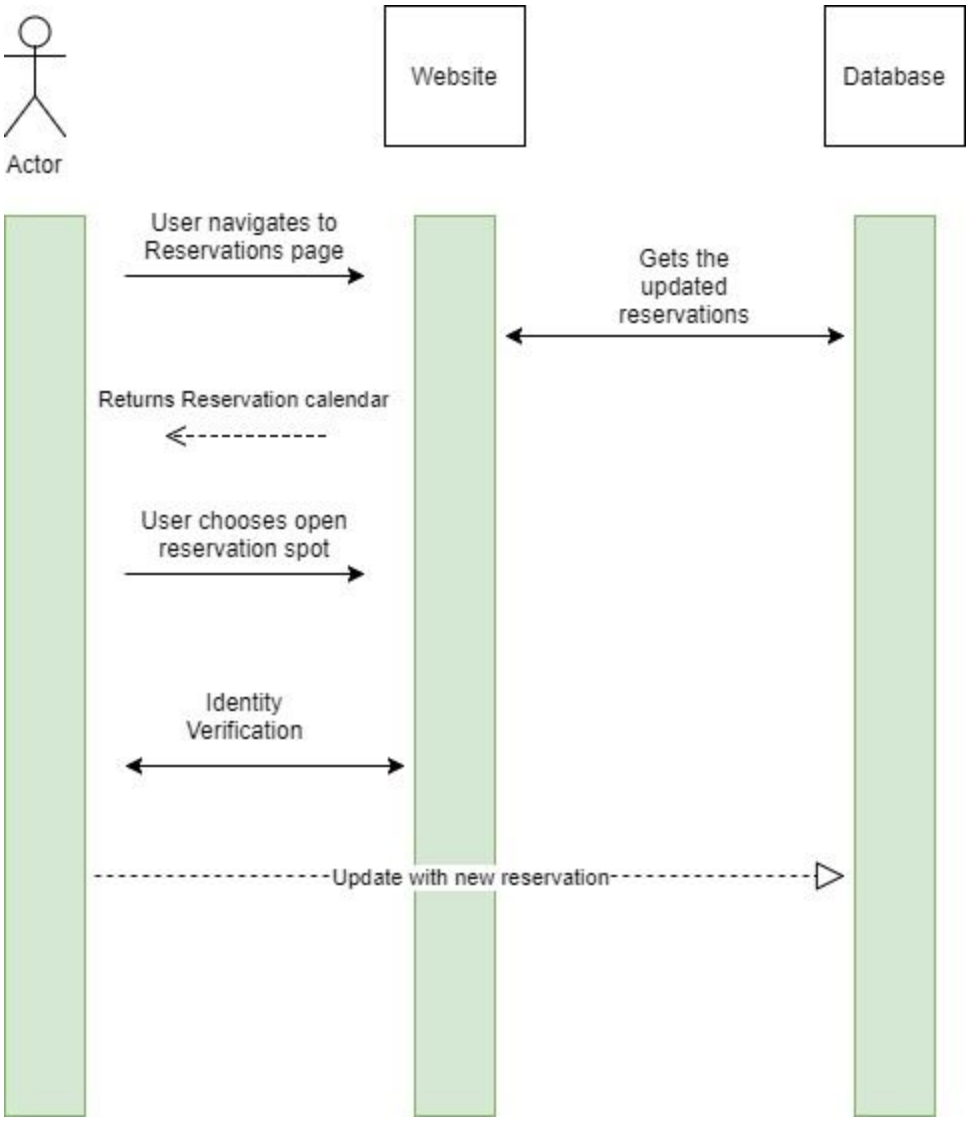
Use Case 2: Serve



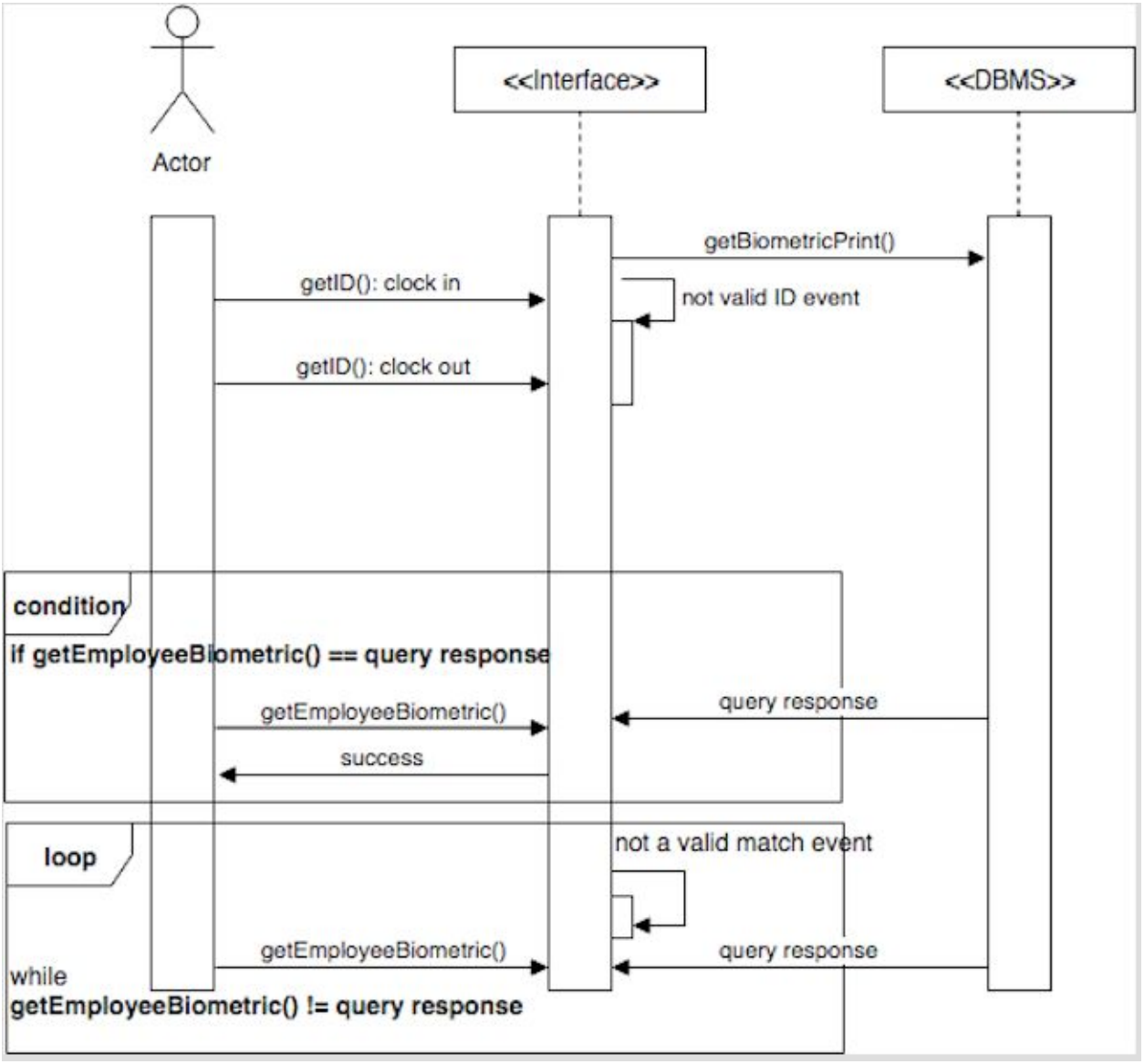
Use Case 3: Seat



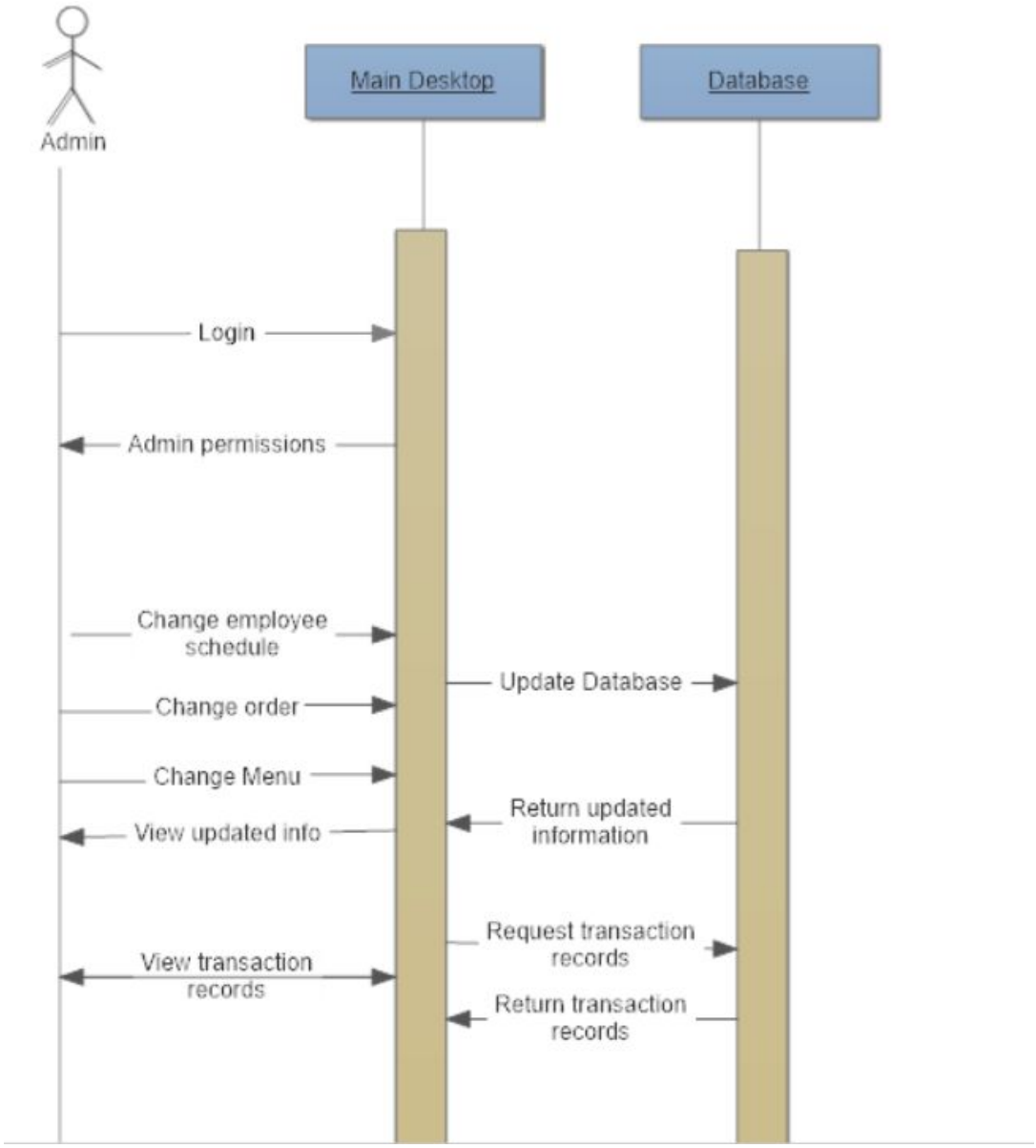
Use Case 4: Reservation



Use Case 5: Clock In/Clock Out

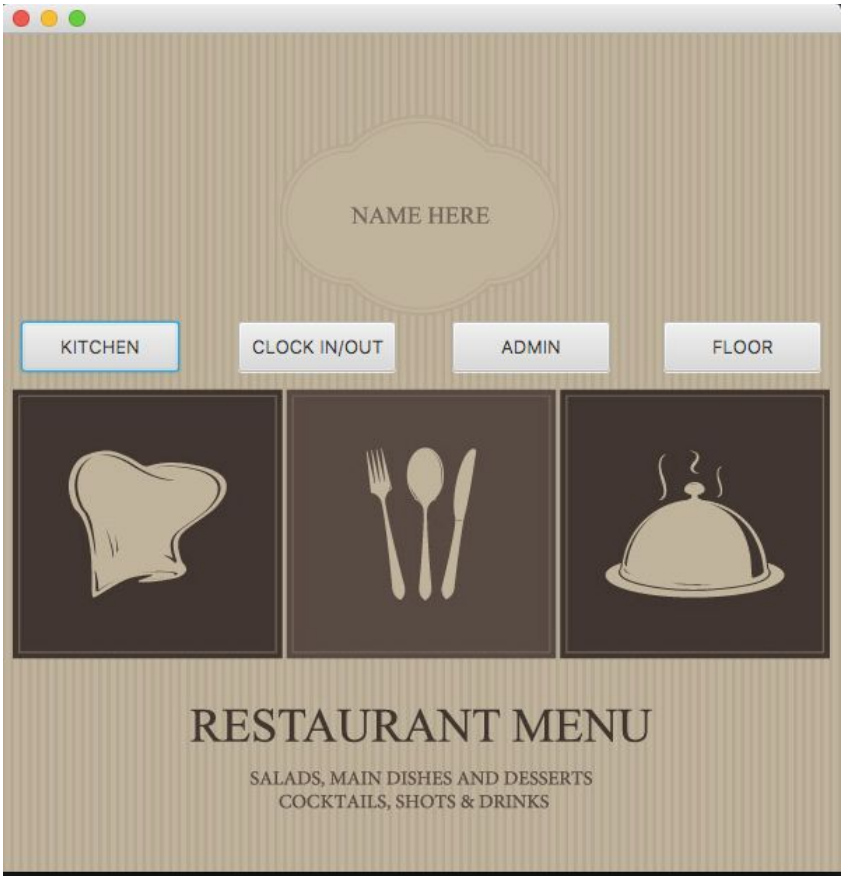


Use Case 6: Admin



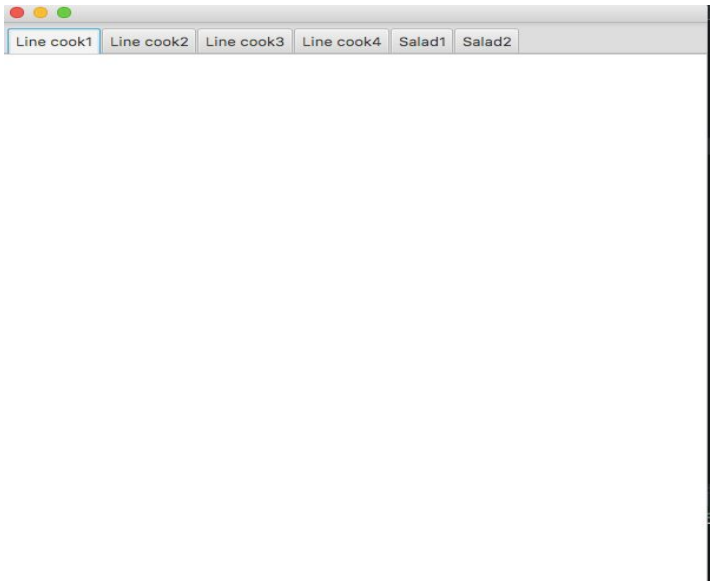
User Interface Specification

Preliminary Design
Restaurant Desktop GUI



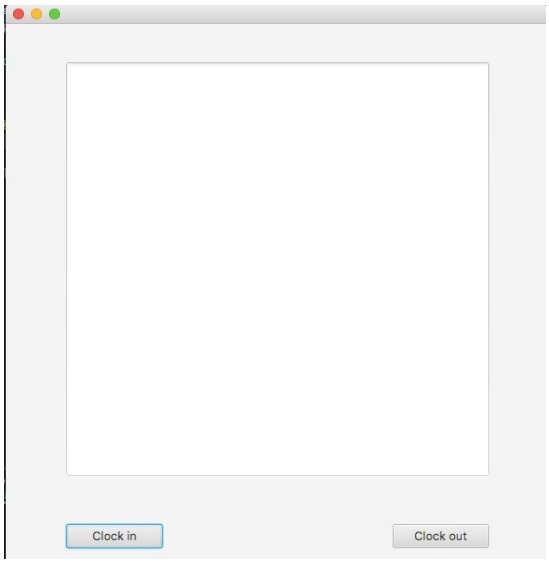
This picture is the current main page of the desktop restaurant automation application. From here the user can either enter the Kitchen, Clock In/Out, Admin, or Floor window.

Kitchen



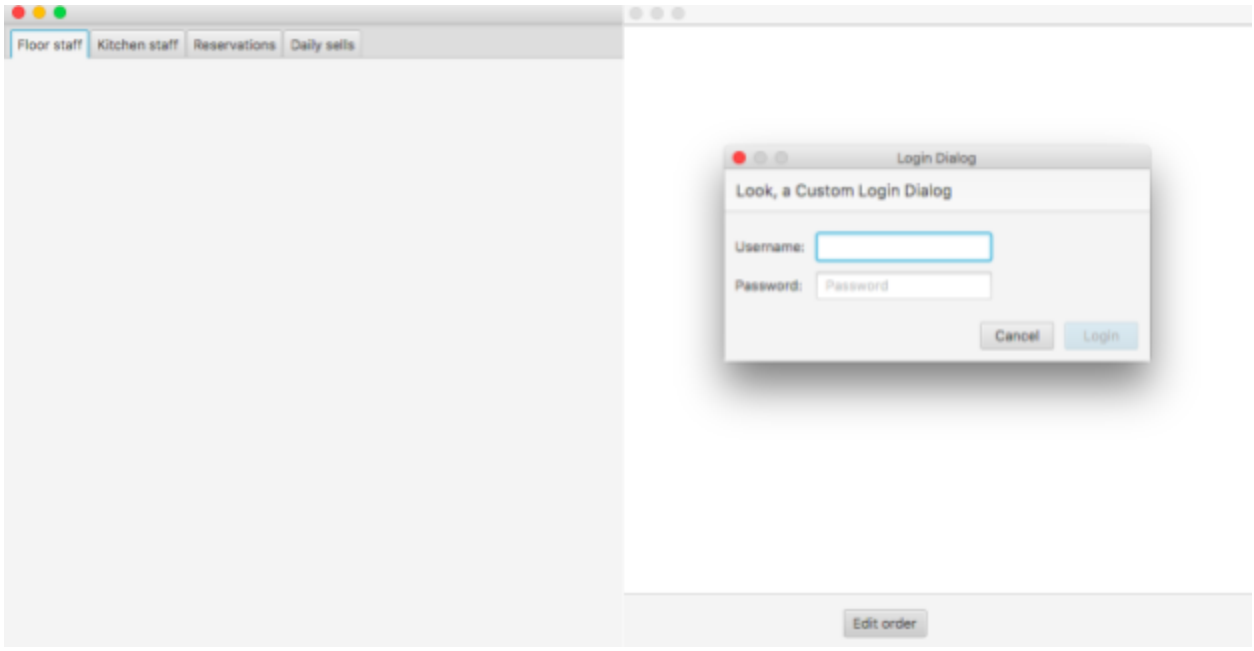
This is the current Kitchen window. Here line cooks have their own tab that will populate with meal orders as their are sent from the waitress.

Clock in/out



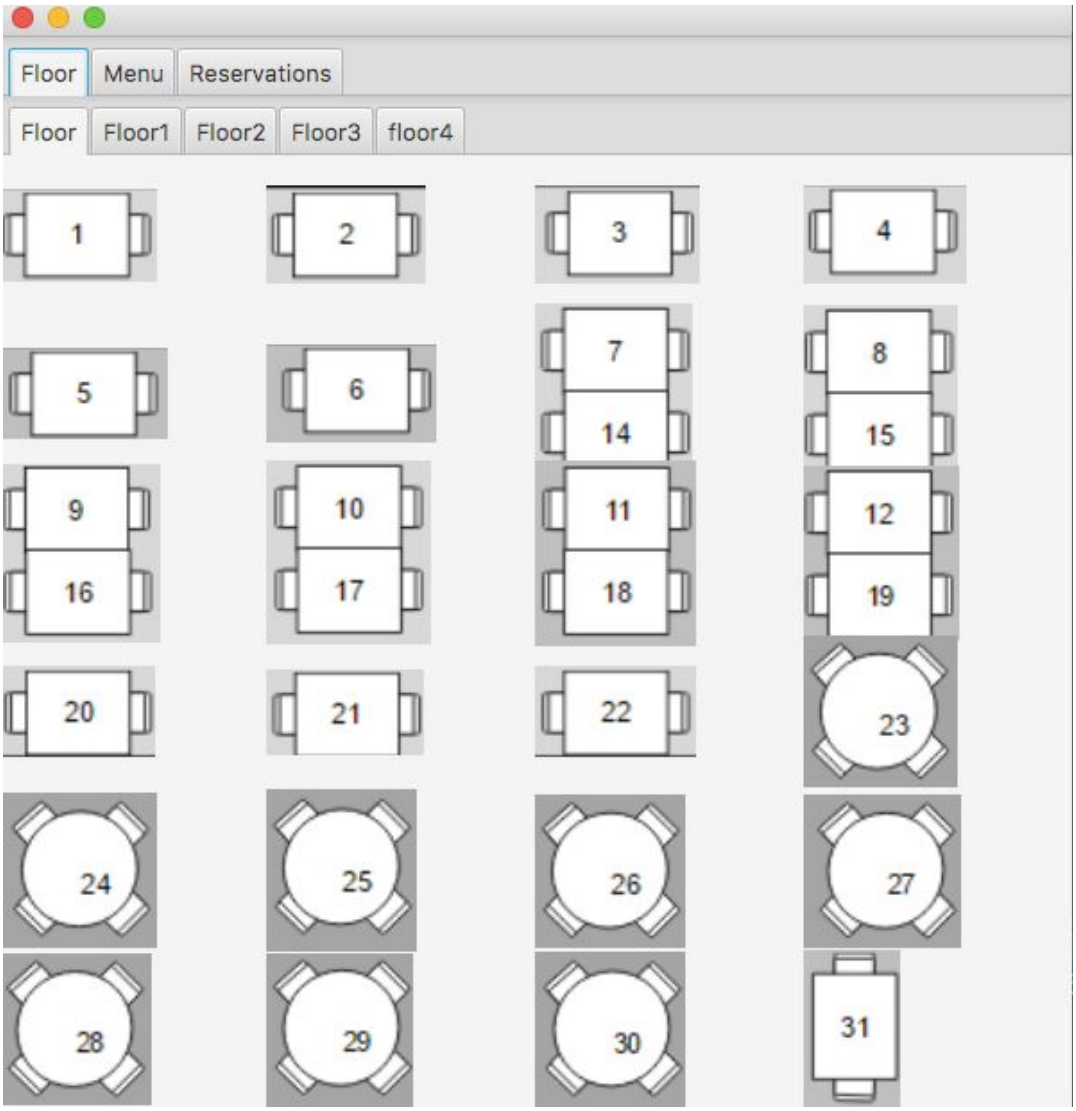
This is the current Clock in/out window. Here employees can log their attendance for work.

Admin



This is the current Admin window. Here users will be prompted to login so that only managers can enter the information and functions available through this page.

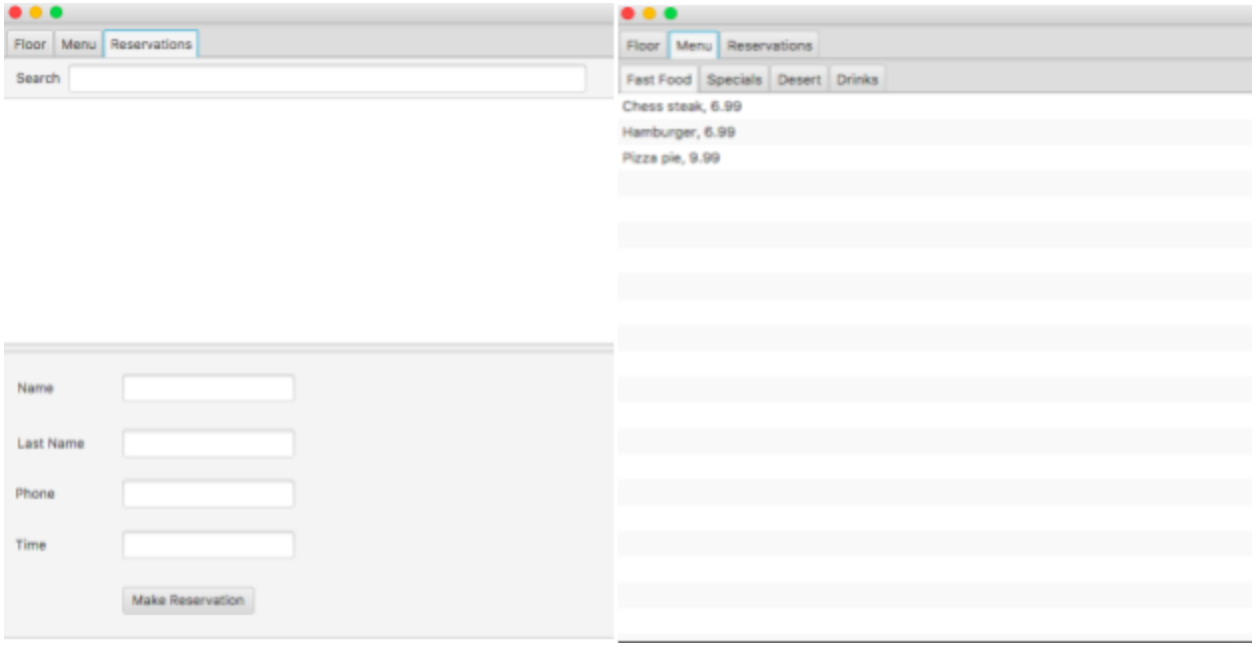
Floor



Here is the Floor menu page. Here users can check the table status of all the tables in the restaurant. In the following images below, green represents ready, blue represents dirt, and red represents occupied.

Table State (green: ready, blue: dirty, red: occupied)

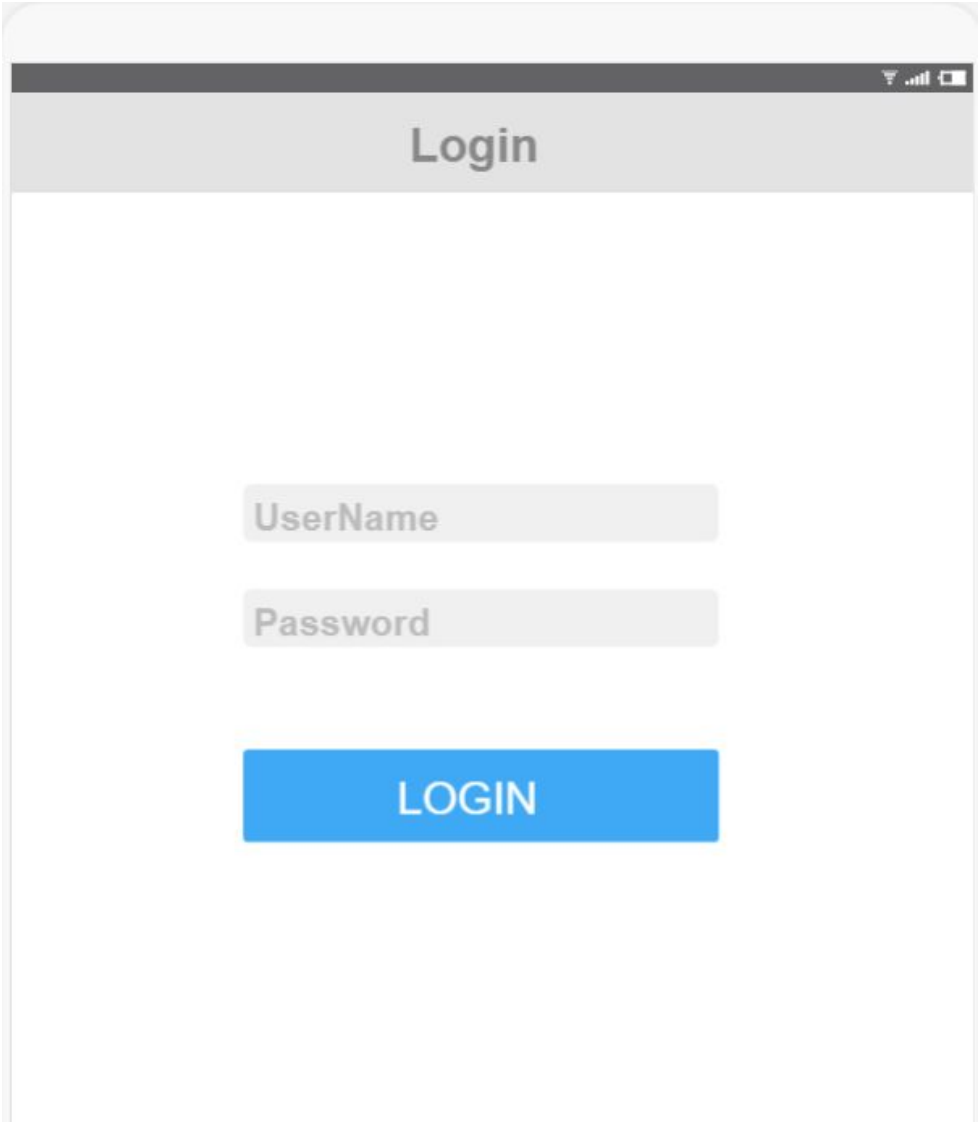
Reservation and Menu Interface.



This is the Reservation and Menu Interface window. For reservations, users can either look for an already made reservation or create a new one. For Menu Interface, users can enter new menu items or update already existing menu items.

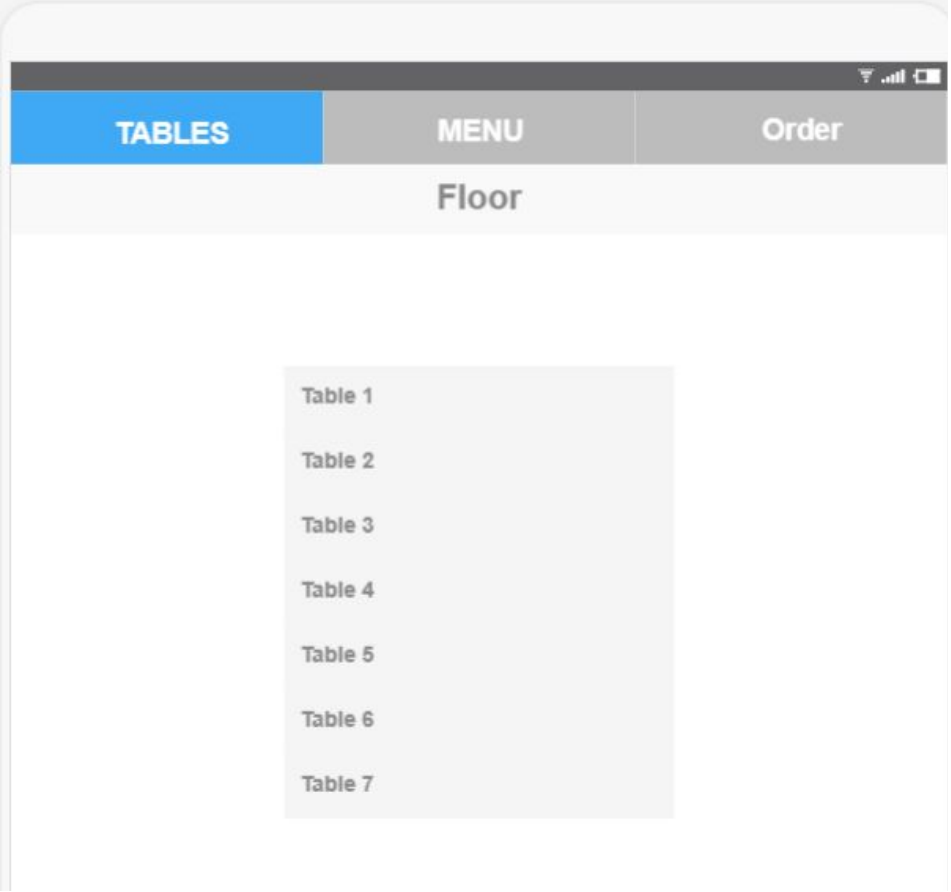
Mobile Application UI Diagrams

Mobile App Login Page



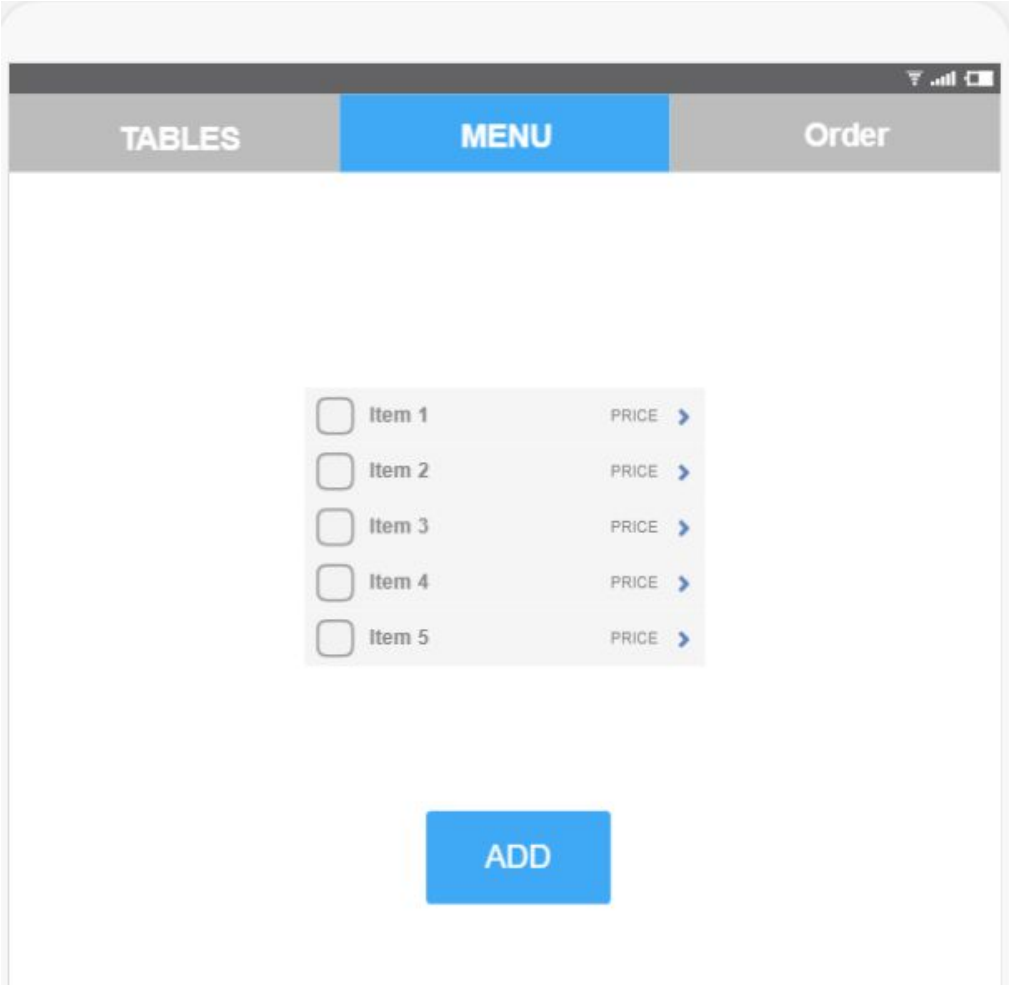
This is the Mobile Application login page. Here Waitresses can login to begin taking orders. By using a login system we can secure the application to only allowing users with access to order items.

Mobile App Tables Page



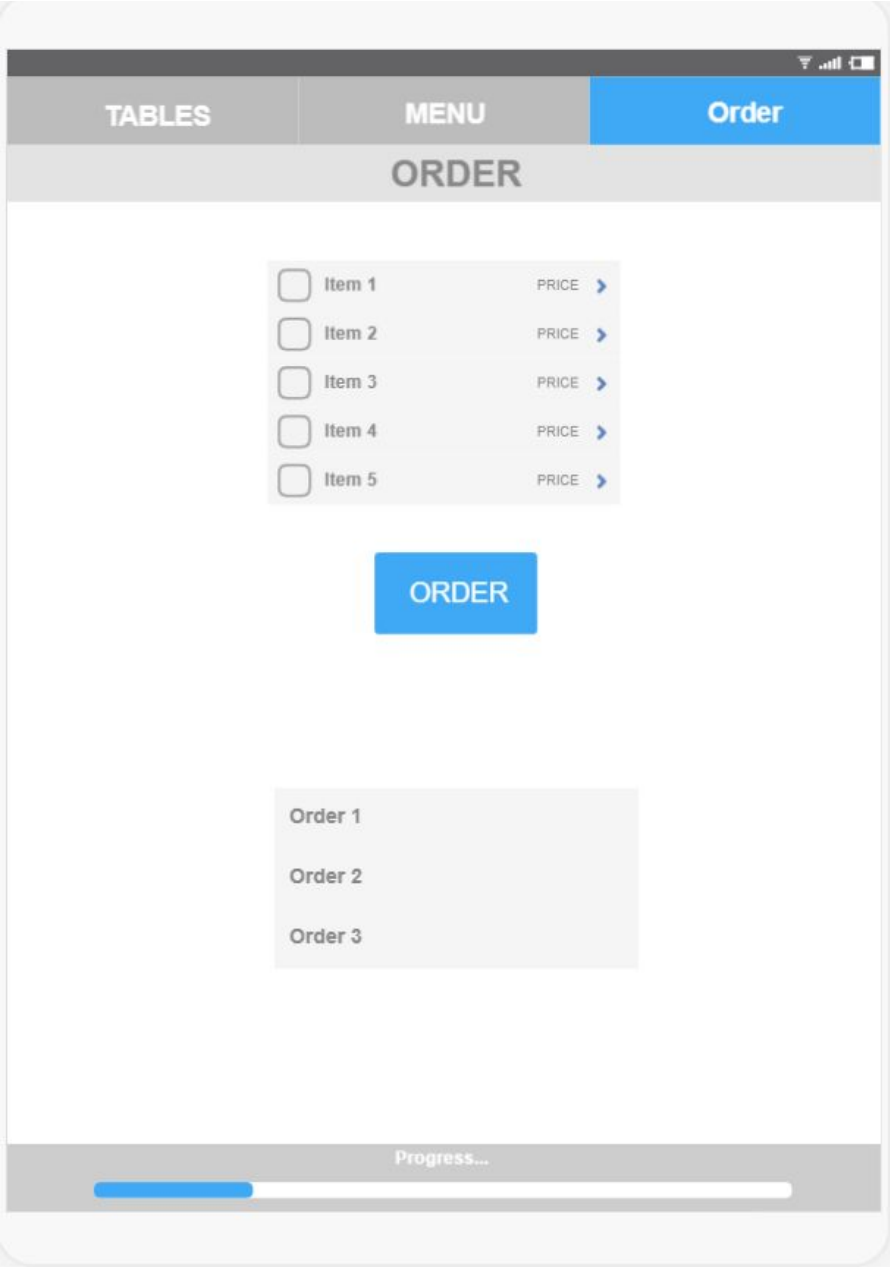
The Mobile App Tables Page will have a drop down list for the floor selection. There users can choose which floor they will be servicing from a drop down list. Under that element is a List View of the tables on the floor. Each table item will be clickable and will lead into the Menu Page.

Mobile App Menu Page



The Menu page has a list view of all the menu items available with the the price on the side. The check box allows the user to select the items and the ADD button will add these items to a order list in the Order page.

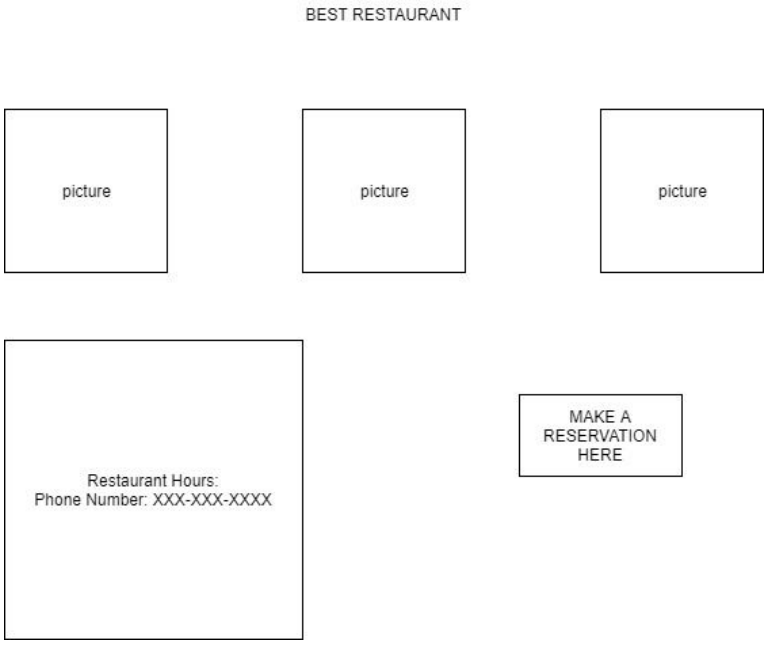
Mobile App Order Page



The Order page allows users to see what items are queued up before sending the order to the kitchen. The top list view will be populated with menu items that were selected and added from the menu page. The ORDER button allows the user to send the order list to the Kitchen electronically. The bottom listview will be populated by the orders already sent to the Kitchen. Clicking on any of the order items from this list will show the progress of it on the bottom of the page through the Progress bar.

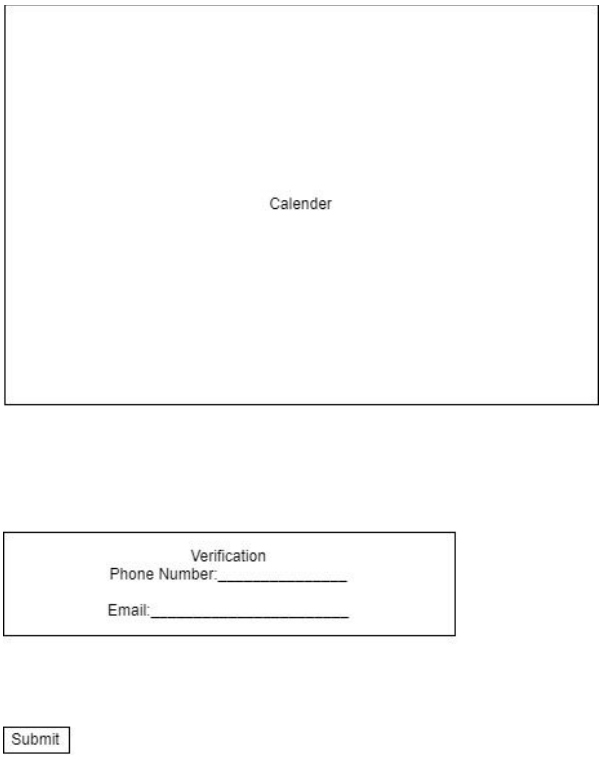
Website

Homepage



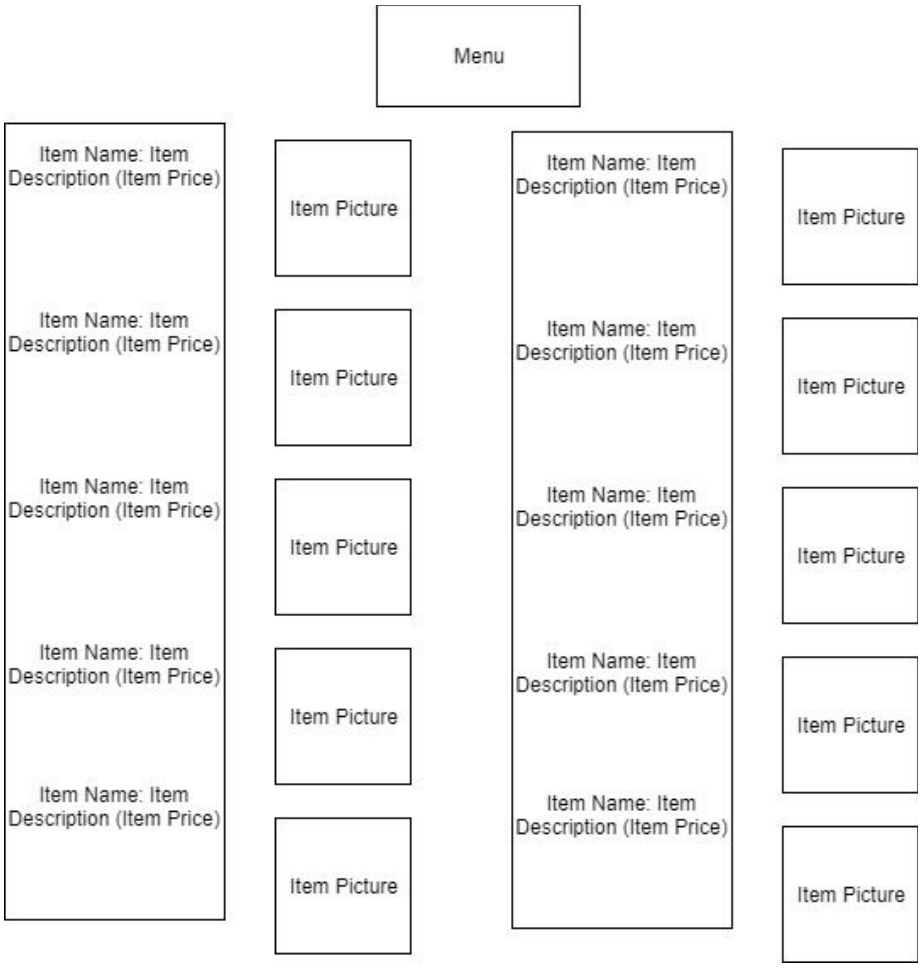
The homepage contains some general information about the restaurant. There are three pictures at the top to showcase some of the restaurant's decor. There are also the restaurant's hours and the phone number. There is a button that directs you to the reservations page.

Reservation Page



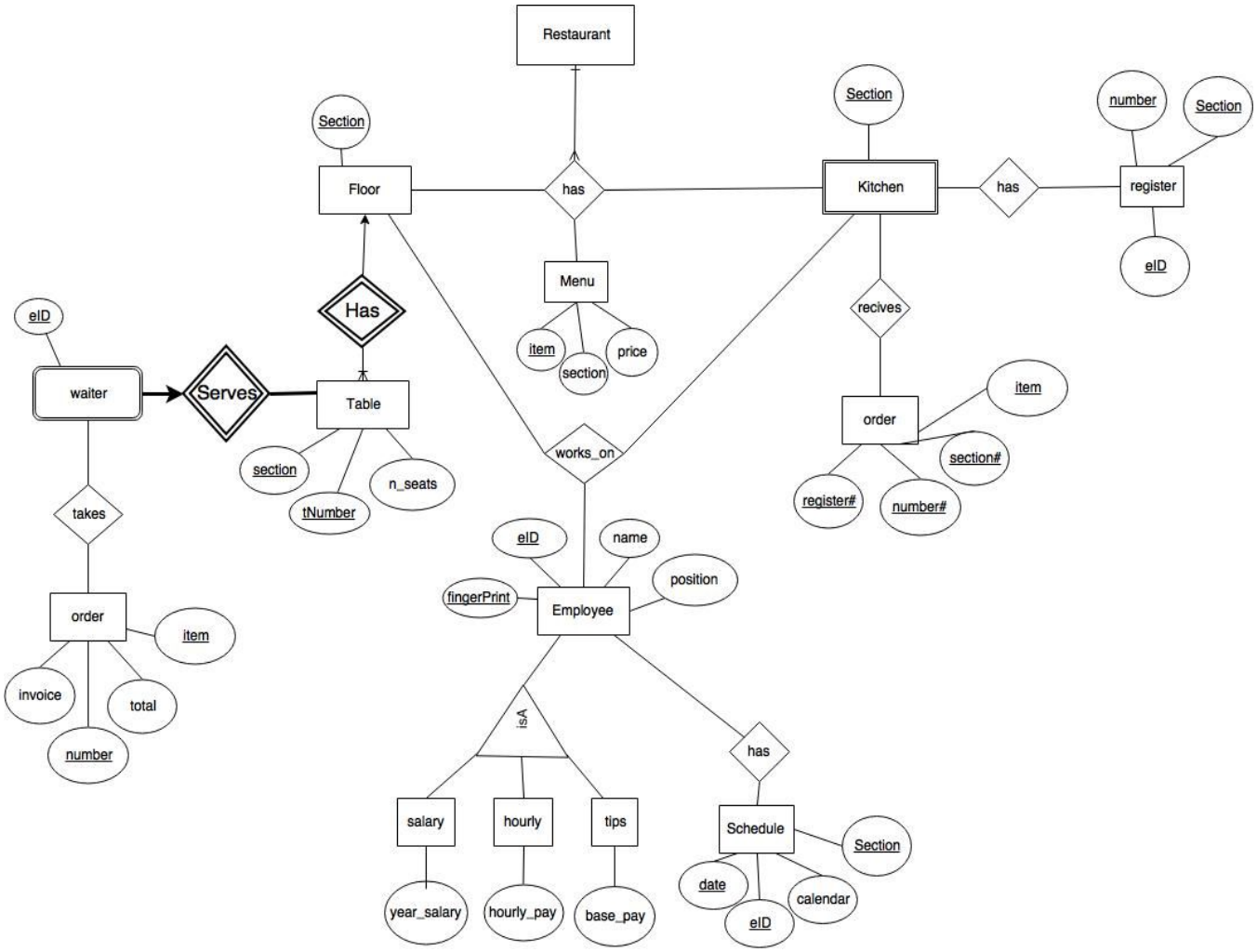
A calendar of all the dates is shown. The availability is shown on the days in the calendar. Then a verification entry is needed, whether phone number or email.

Menu



The menu page contains all the menu items with their descriptions, name, price, and picture.

Restaurant Automation ER diagram



This is the entity relation database system diagram created for the Restaurant Automation System. Here you can see the classes, functions, objects and how they work together.

User Effort Estimation

Scenario 1: Waiter Places Order

1. Navigation: From main activity press Menu button (3 taps minimum)
    a. Scroll up and down to find desired item
    b. Tap once to add to order, tap again to add another
    c. Once all desired items are added to order, press Order button to send to kitchen

Scenario 2: Host Seats Patrons

1. Navigation: From main window, enter table status tab (3 taps)
    a. Search through floors for available tables by pressing floor tabs
    b. Select table by clicking on desired table and setting name to patron's name
    c. Select an available waiter from list to service patron's party

Scenario 3: Busboy Cleans Table

1. Navigation: Select the option to look at the table view (3 taps)
    a. Waiter/busboy selects a table that has status called "needs to be cleaned" and clicks it to change the status to "cleaning table"
    b. Once the waiter/busboy is done cleaning the table, the waiter/busboy clicks the same table again to change the status to "ready table"

Scenario 4: Chef Prepares Order

1. Navigation: Select the option to go to the item queue (4 taps)
    a. Chef chooses an order from the list to start preparing. The queue is listed in order of when the item was inputted by a waiter, alongside all appropriate ingredients for the items (depending if customer changes ingredients)
    b. Chef selects "begin preparing" option and starts to prepare the order
    c. Chef selects "complete" option when the order is ready

Scenario 5: Customer makes reservation online

1. Navigation: Select the "Make Reservation" button (3 clicks, 3 info entries)
    a. Customer clicks the reservation page button
    b. Customer enters their information
    c. Customer presses submit

Unadjusted Use Case Points:

Use Case Weight

Type	**Description**	**Weight Amount**
Simple	One participating actor. 3 or less concepts	5
Average	Two or more participating actors 4 concepts	10
Complex	Three or more participating actors 4+ concepts	15

Use Case	**Use Case Weight**
UC-1: Order	15
UC-2: Serve	10
UC-2: Seat	10
UC-4: Reservation	5
UC-5: Clock in/out	5
UC-6: Admin	15

UUCP = 60

Technical Complexity Factor:

Technical Complexity Factor	Technical Complexity Factor Weight	TCF perceived complexity	Complexity Factor
1 - Distributed System	2	4	8
2 - Performance	1	4	4
3 - Ease of use	.5	5	2.5
4 - User Efficiency	1	4	4
5 - Ease of install	.5	3	1.5
6 - Reusability	1	0	0
7 - Portability	2	2	4

TCF = 0.6 + 0.01(Total Complexity Factor)

TCF = 0.6 + .24

TCF = 0.84

Environmental Factor:

Environmental factor	Weight	Perceived Impact	Complexity Factor
1 - System Familiarity	1.5	4	6
2 - Work Experience	.5	5	2.5
3 - Motivation	1	3	3
4 -	2	1	2
5 - Part-time staff	-.5	3	-1.5
6 - Language barrier	-1	2	-2

ECF = 1.4 + -.03*(Total Environmental Complexity Factor)

ECF = 1.4 + -.03*(10)

ECF = 1.1

User Complexity Points & Duration

UCP = UUCP * TCF * ECF

UCP = 60 * 0.84 * 1.1

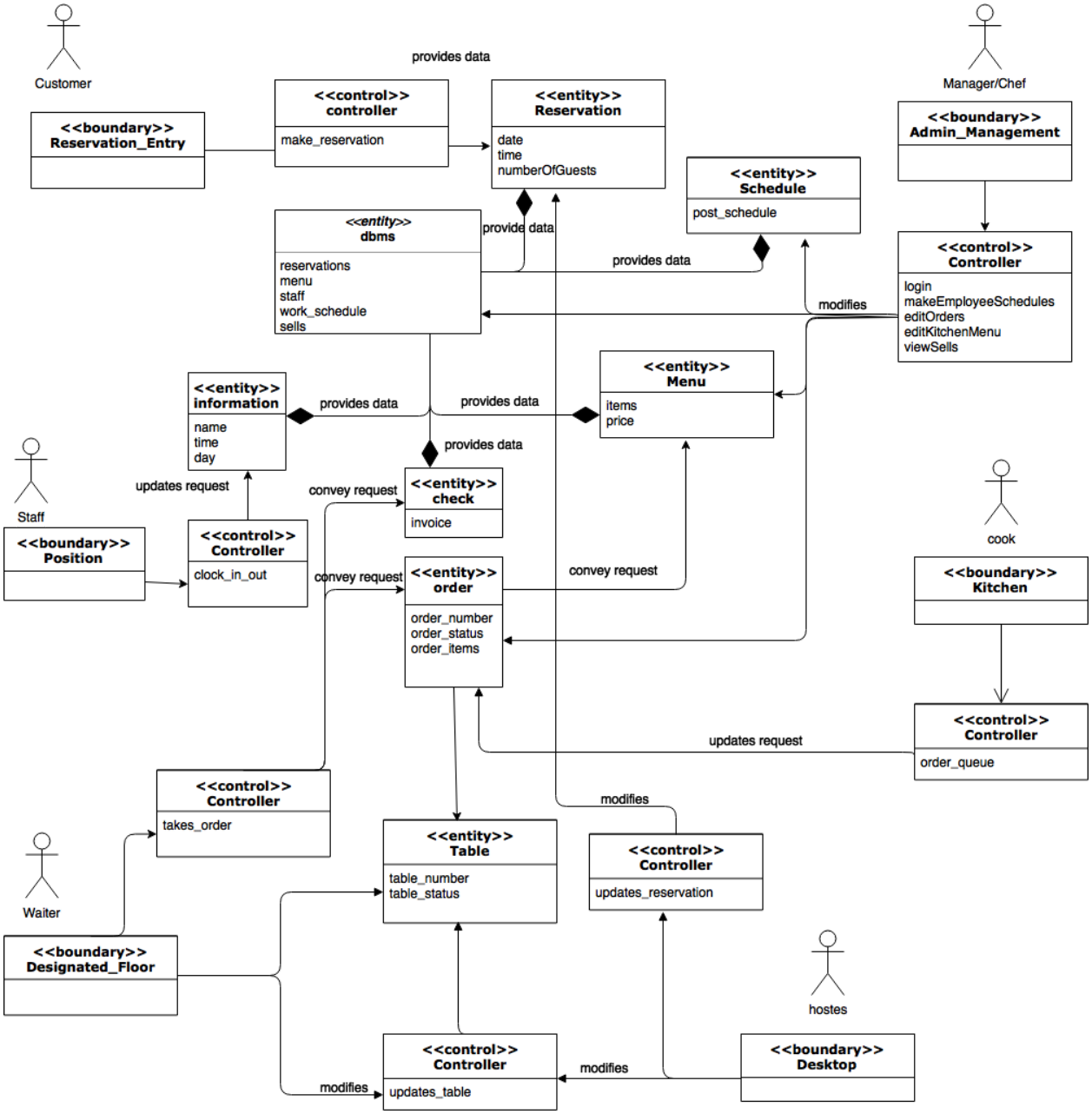
UCP = 55.44 = 55 use case points

Duration = UCP * Productivity Factor

Duration = 55 * 28 = 1540 Hours

Domain Analysis

Domain Model:



Concept Definitions:

C	Concept	Type	Responsibility Description
C1	Employee	D	Staff working at the restaurant that is registered to the system
C2	Manager	D	Manages the employees time schedules and work tasks
C3	Host/Hostess	D	Greets and seats the guests at their tables. Manages the incoming reservations as well.
C4	Waiter/Waitress	D	Takes orders from customers and sends them in
C5	Kitchen	D	Prepares food and notifies waters when orders are ready
C6	Floor	K	System shows the floor occupancy
C7	Table	D	Waiter assigns to guests when occupied
C8	Menu	D	Displays the food options
C9	MenuItem	D	Kitchen updates the items in the menu
C10	tableStatus	K	System updates status when being used by guests
C11	MenuModifier	D	
C12	Reservation	D	A customer makes a reservation to assure a table at an agreed upon time.
C13	ReservationCalendar	K	Keeps track of all reservations
C14	ReservationModifier	D	Modify the reservation calendar
C15	OrderItem	D	System sends order to kitchen
C16	OrderQueue	K	Keeps track of all the unfinished orders
C17	Check	K	Keeps track of all items ordered at a table with associated price, subtotal, total, and calculated tips
C18	CheckModifier	D	Allows user to change any of the check's associated price, subtotal, total, and calculated tips. Allows to add and remove items from check
C19	Users	K	Keeps track of all employee users, passwords, and permissions

C20	UserChecker	D	Checks if the given username and password are valid, provides entryway for employees to system
C21	Clock	K	Displays current day and time for employees to check shifts
C22	ClockSchedule	K	Displays all weekly employee schedules with associated dates and times
C23	ClockIn	K	Keeps track of when employees start and stop work
C24	WebPageController	D	Central intermediary between InferfacePage and PageMaker, and all reservation web-page actions
C25	PageMaker	D	Collects information needed to create page interface
C26	InterfacePage	D	Acts as the GUI communicator between the user (customer) and system

ii) Association Definitions:

Concept Pair:	**Association Description:**	**Association:**
Manager ⇔ ClockSchedule	Manager assigns work schedule employees	Modifies
ClockIn ⇔ Employee	Provides data to system when employees clock in to work	Provides data
Manager ⇔ MenuModifier	Manager updates menu	Modifies
Host/Hostess ⇔ Customer	Host seats customer to unoccupied table	Conveys Requests
Host/Hostess ⇔ Reservation	Host takes reservations from customers to assign tables on date	Conveys Requests
Host/Hostess ⇔ Floor	Host updates the floor occupancy	Provides Data
Host/Hostess ⇔ Table	Host updates the table status when reserving/assigning table	Provides data
Customer ⇔ Menu	Customer gets information from the menu	Provides data
Customer ⇔ Waiter/Waitress	Customer gives waiter/waitress order information	Conveys request

Customer ⇔ Website	Customer receives restaurant information	Provides Data
Customer ⇔ Reservation Modifier	Customer can book and edit their own reservation	Conveys request
Waiter ⇔ Kitchen	Waiter relays orders to Kitchen	Conveys request
Waiter ⇔ OrderQueue	Waiter checks for updates from order queue	Requests updates
Kitchen ⇔ OrderQueue	Kitchen updates the order status for the order queue	Modifies
Waiter ⇔ CheckModifier	Waiter updates the check of orders into the system	Modifies/Provides Data
Employee ⇔ Clock	Employee informs the schedule of clock in/clock out times	Provides Data

iii) Attribute Definitions:

Category:	**Attribute:**	**Description:**
Employee	userID	String representing each individual employee in the software suite
Employee	password	String stored and associated with each userID
Employee	shiftStart	Time when employee starts shift
Employee	shiftEnd	Time when employee ends shift
Manager	managerName	Name of manager
Manager	managerID	Unique number representing a manager
Manager	managerPermissions	Manager permissions include: changing orders, posting staff schedules, seeing daily transaction reports, log-in time of employees, all waiter permissions
Host/Hostess	hostName	Name of host/hostess

Host/Hostess	hostID	Unique number representing a host/hostess
Host/Hostess	hostPermissions	Host permissions include: creating reservations, updating floor occupancy, modifying table status.
Waiter	waiterName	Name of waiter
Waiter	waiterID	Unique number representing a waiter
Waiter	waiterPermissions	Waiter permissions include: seeing which tables associated with, see status of tables
Waiter	tableAssociation	Bitmap showing all tables assigned to waiter
Floor	floorLevel	Assigns each floor a level to display the array of tables
Floor	floorStatus	Shows the occupancy percentage of each floor inside the restaurant
Table	freeTables	Array of all available tables open for customers in queue for a table
Table	occupiedTables	Array of all tables in use by a customer party, cannot be given to queued customer
Table	tableStatus	Shows if a current table is: in use, waiting for order, eating, waiting for check, needs cleaning, is ready
MenuItem	itemName	Menu item's name
MenuItem	itemDescription	Menu item's description
MenuItem	itemPrice	Menu item's price
Order	order	Order of a party at a table, list of items for chef to prepare

Order	orderID	Unique number representing table's order
Order	isOrderReady	Boolean showing status of order
OrderQueue	orderQueue	List of all orders needed to be prepared by chefs
Check	checkID	Unique number representing table's check
Check	checkSubtotal	Overall subtotal table has to pay on the bill
Check	checkTotal	Overall total after tax table has to pay on bill
Check	checkTip	Calculated possible tips after total for: 15%, 18%, 20% tips
TransactionReport	Day	Signifies transaction report is daily
TransactionReport	totalGain	Sum of all methods of revenue for the day
TransactionReport	totalLoss	Sum of all methods of losses for the day
TransactionReport	netGain	Difference of the gains and losses
Schedule	employeeName	Name of employee
Schedule	date	All of the dates specified employee is working
Schedule	time	Time frames specified for each date employee is working

Traceability Matrix:

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18	C19	C20	C21	C22	C23
UC1	*			*	*	*	*	*	*						*								
UC2	*			*	*	*	*			*						*	*						
UC3	*		*	*		*	*			*		*											
UC4		*				*				*		*	*	*									
UC5	*	*	*	*	*																*	*	*
UC6		*									*			*				*	*	*		*	*

System Operation Contracts

Name	Customer Places Order
Responsibilities	Customers place their meal orders through the waiters
Use Cases	UC-1
Exception	None
Preconditions	Customers are already seated and Waiters are logged into the mobile app.
Postconditions	The order is placed and sent to the Kitchen, status will be updated on mobile app.

Name	Manage Meal Orders
Responsibilities	Kitchen staff will prepare placed orders and notify waiters when to serve
Use Cases	UC-2
Exception	None
Preconditions	Meal orders have already been placed through the mobile app
Postconditions	On the Mobile app, waiters can see the change of progress of orders, thus knowing when to pick up and serve ready meals

Name	Logging Work Hours
Responsibilities	Restaurant Staff are able to clock in and out their work hours using the desktop app
Use Cases	UC-5
Exception	None
Preconditions	Users are employees and logged in to desktop application
Postconditions	After clocking out, the total time worked is saved in the database

Name	Seating Customers
Responsibilities	As customers come in, host are able to find appropriate seating for them
Use Cases	UC-3
Exception	None
Preconditions	Host is logged in to desktop app and viewing the floor page
Postconditions	Host leads party to table and hands off responsibility to the waiter of the floor to serve them

Name	Reservation Management
Responsibilities	Hosts can create or verify customers reservations
Use Cases	UC-4
Exception	None
Preconditions	Hosts are logged into the desktop app and viewing the Reservations page
Postconditions	Through the reservation page, hosts can verify name of customer to see if they have a reservation or create a new one for a future visit

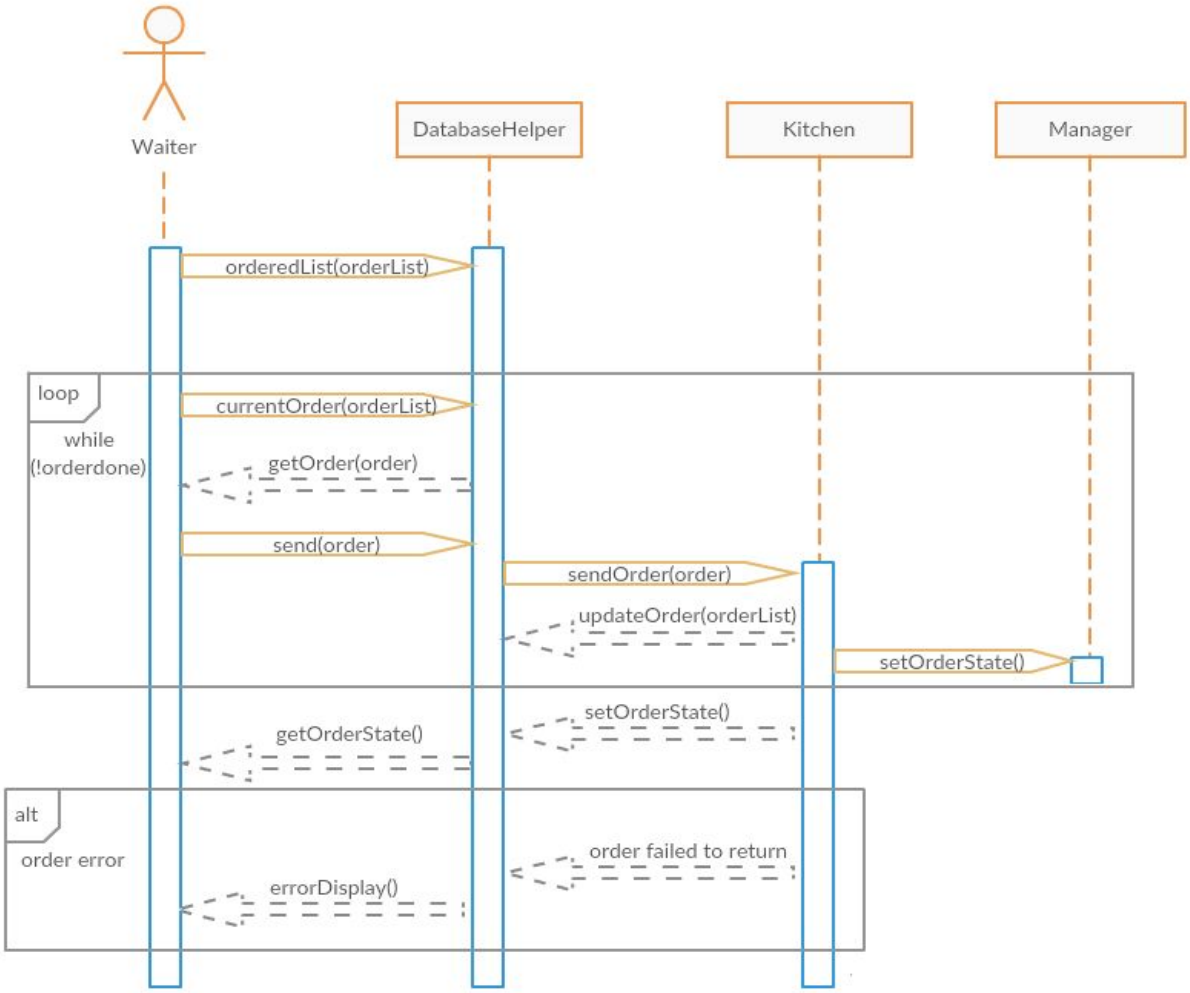
Name	Manage Menu Items
Responsibilities	User can update the Menu either adding, deleting or updating menu items
Use Cases	UC-6
Exception	None
Preconditions	User is an admin/chef logged into the desktop app and viewing the admin page
Postconditions	Menu items are updated with user's modification

Name	View Worker Hours
Responsibilities	Check all worker hours
Use Cases	UC-6
Exception	None
Preconditions	Admins are logged into desktop application and viewing Admin page
Postconditions	View logs of hours worked by each employee

Name	View Receipts
Responsibilities	View a log of all receipts of customers
Use Cases	UC-6
Exception	None
Preconditions	Admins are logged into desktop application and viewing Admin page
Postconditions	View logs of all receipts made

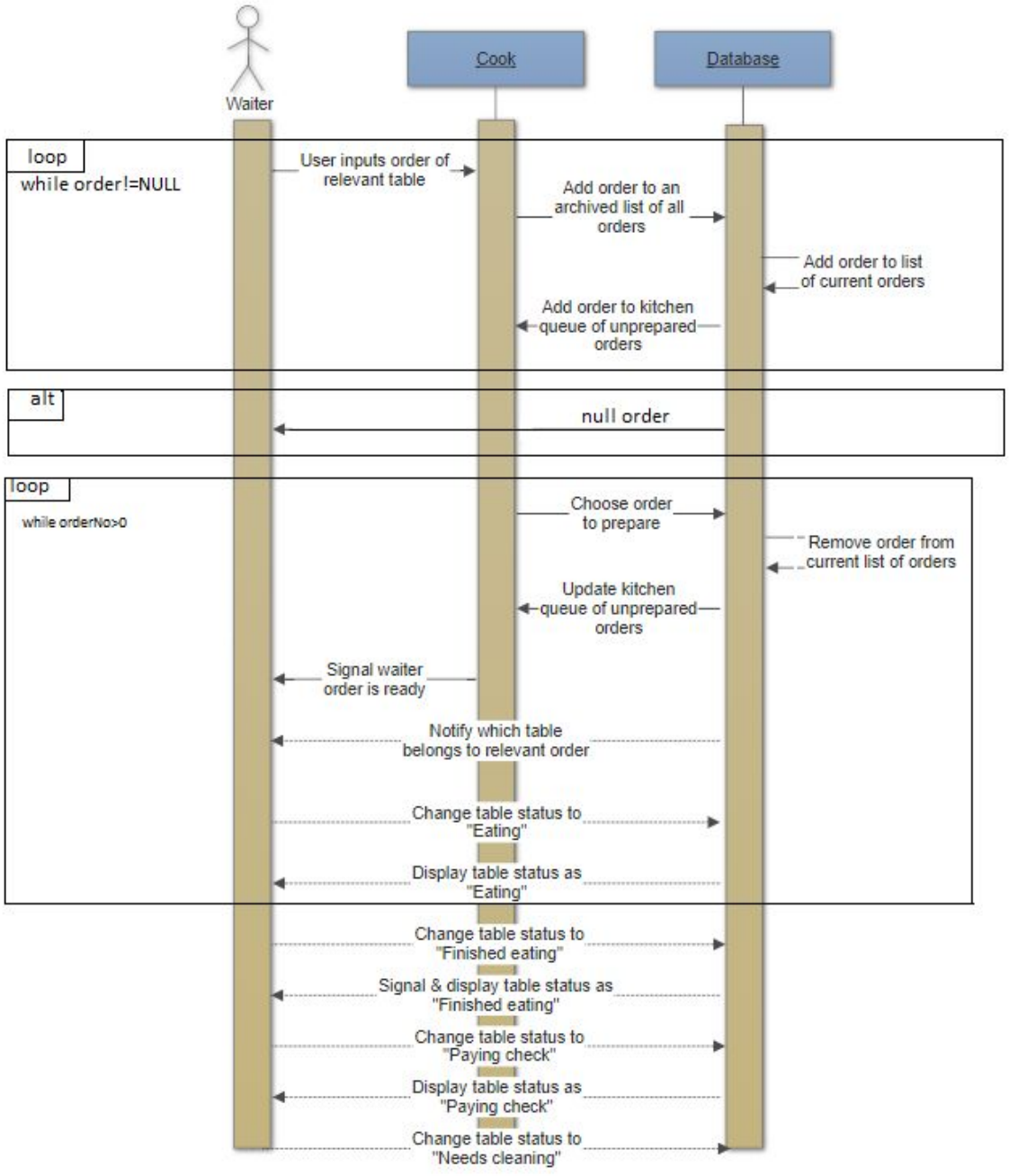
Interaction diagrams

Use Case 1: Order



In this Use Case, the waiter begins by placing an order using the mobile device. With OrderedList the waiter is enabled to view the orders from the database. The waiter then selects the currentOrder and retrieves the current order's list. The database returns the order via getOrder so the waiter can then add an order with send(order). The order is sent to the kitchen with sendOrder and the chef may then update the orderlist to the database along with the orderstatus to the manager. This is looped until the order is done. Once the order is completed, the order status is sent to the database and retrieved by the waiter. If the order fails to return, an error is displayed to the waiter to be handled in person.

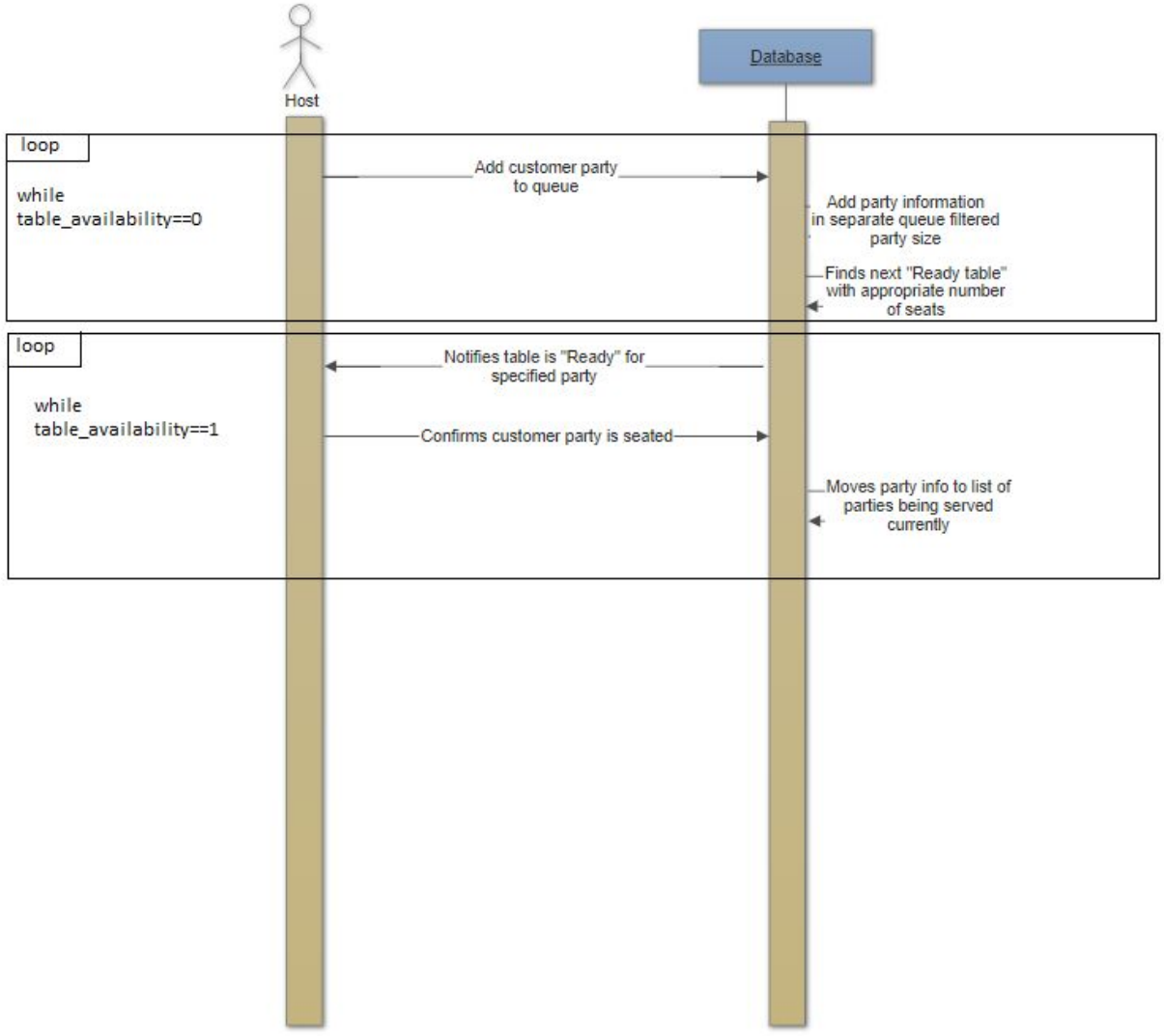
Use Case 2: Serve



In this use case, the waiter is notified when the cook has finished an order and is ready to send the order to the table. The system has a database that keeps track of all of the orders in a queue and subsequently notifies the waiter when an order is ready, thus dequeuing it. When the order is served to the respective table, the table's status is then changed to "eating." Afterwards, once the

table is finished with their meals, the table's status is changed to "finished eating" and the waiter is notified to collect the check. Finally, once the check is paid to the system, the table's status is changed to "needs cleaning" to set up for the next customer(s). In the main success case for an authorised user, the system will accurately assess this data over time and keep a precise check on each table's status.

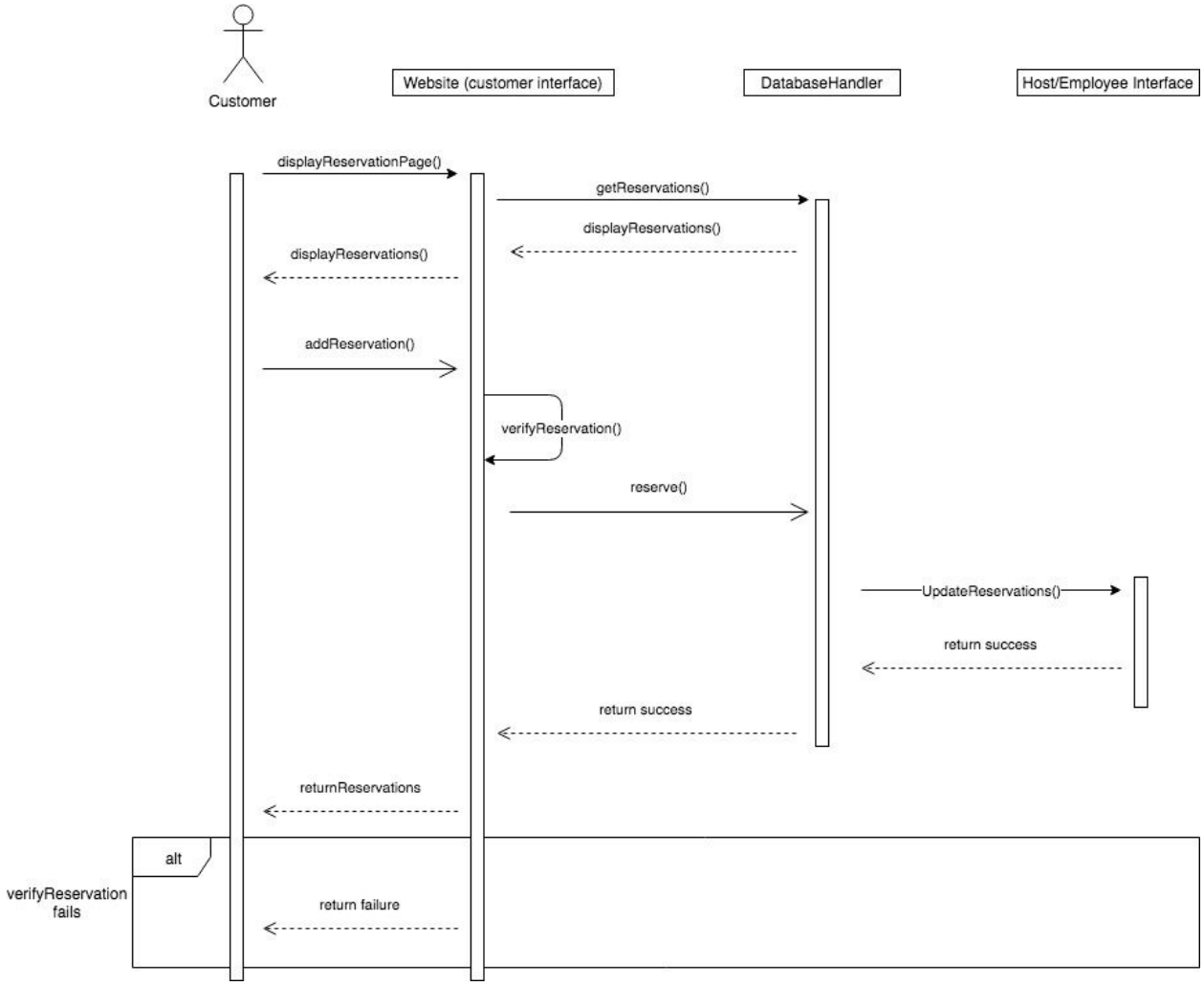
Use Case 3: Seat



For the seating use case, the customer enters the restaurant and selects the 'seating' option. The host adds the customer to a seating queue on the database. The party size is calculated with the current seating options to find an optimal available table. When the table is

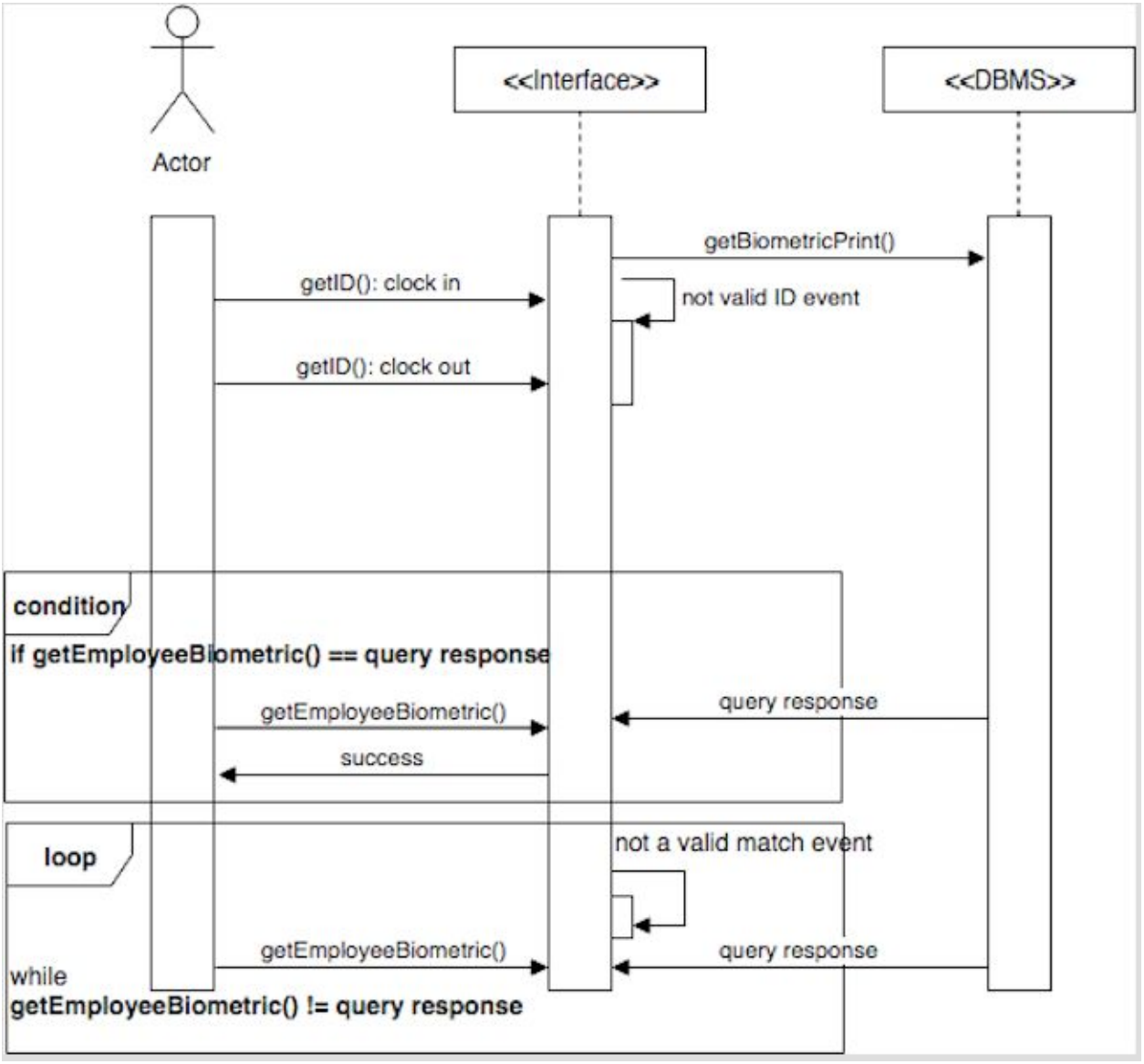
open, the database notifies the customer that they are ready to be seated. The host directs the party to the table and confirms to the database that the customer is seated. The database then moves the customers to a list of parties currently being served.

Use Case 4: Reservation



In this use case, there are two type of users that may interact with reservation UI, customer and staff member (hostess or manager). Reservation navigation may be access through restaurant website for customers and also via desktop by restaurant staff. All information given by the customer will be stored in the database server. Once reservation is made, user will get a confirmation. In case a user might need help on a existent reservation, a staff member can access this information via desktop queries to the database server. In short, reservation UI gives users access to secured data in the database system for management purposes.

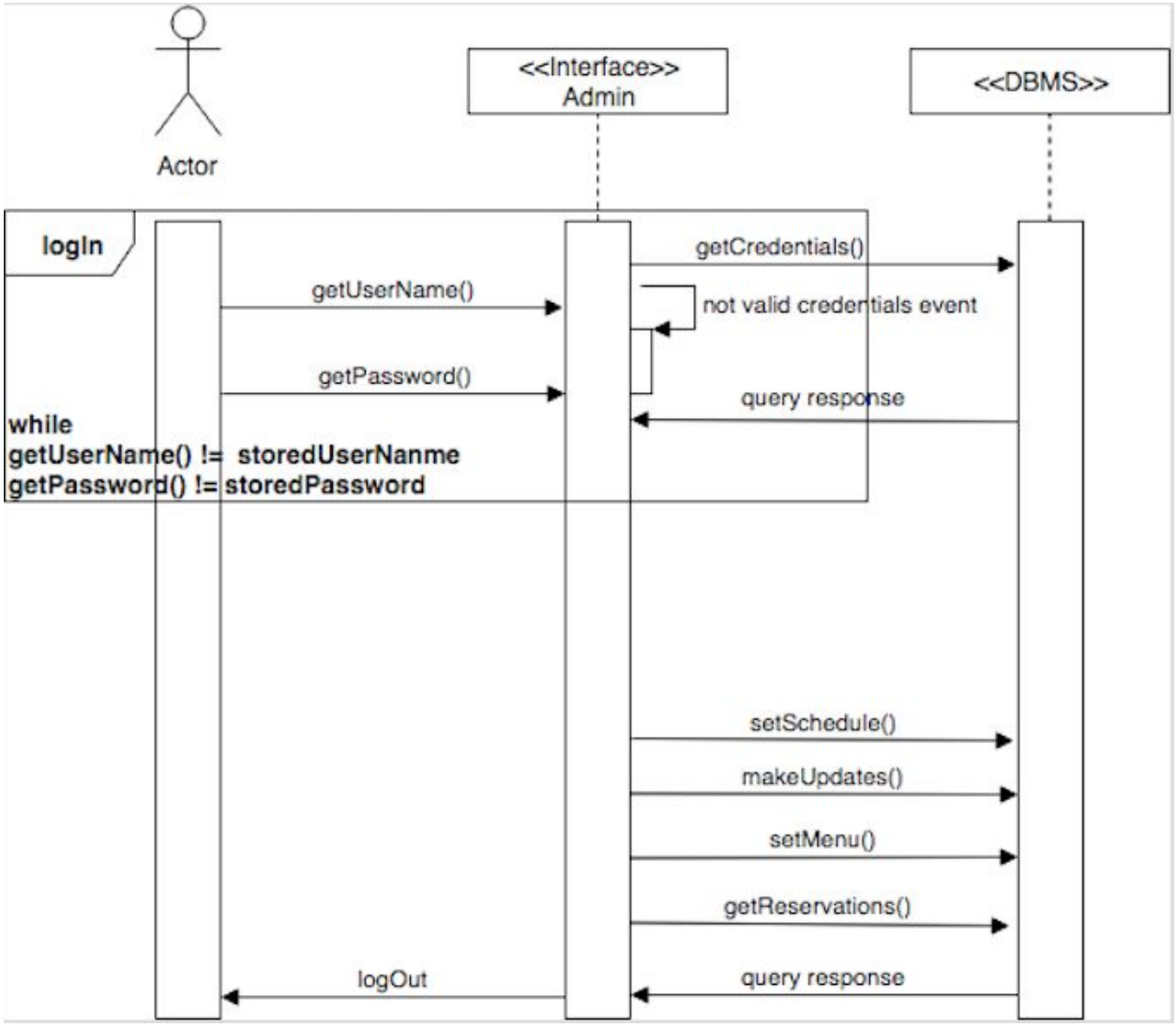
Use Case 5: Clock In/Clock Out



User gets prompt to enter employee ID before pressing clock in/out button. If no/invalid id, program will continue to ask without moving to a new state. Once valid ID number entered then the program will trigger a query to the DBMS in order to get users credentials including biometric print stored. Once all credentials are pulled from the DBMS, then the user once again will be prompt to enter a validation condition, his/her finger into the scanner device in order to get a sample biometric. Once sample is collected, this will be compared to the one stored in DB

for validation. If match, then clock in/out successfully completed, loop until valid match is entered or transaction is cancelled.

Use Case 6: Admin

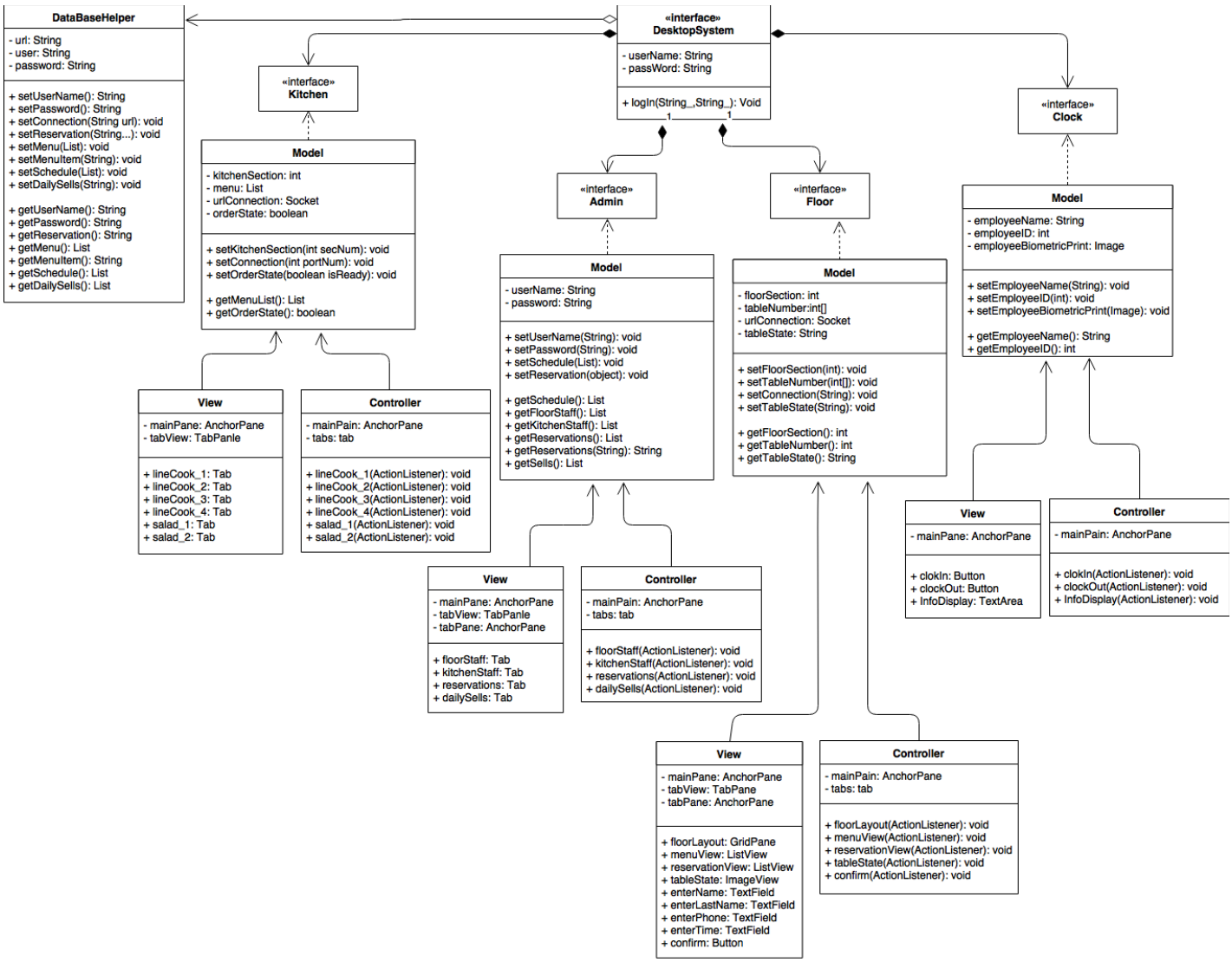


User gets prompt to enter user name and password ID before pressing clock in/out button. If invalid credentials, program will continue to ask three more time before locking user out. Once valid credentials, the program will trigger a query to the DBMS in order to get users credentials stored in database. Once all credentials are pulled from the DBMS, then the user will get access to the system and allowed to make any desire changes.

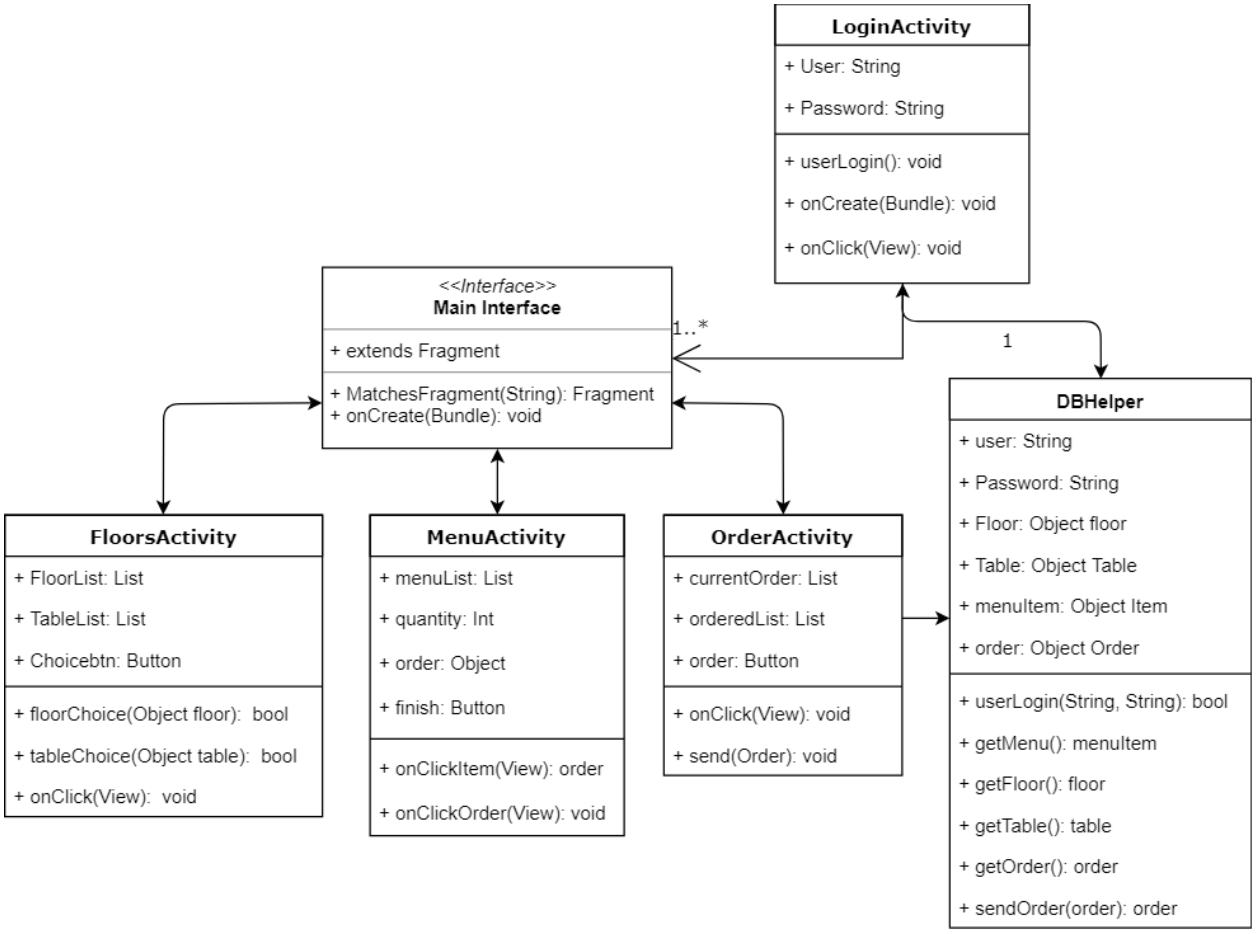
Class Diagram and Interface Specification

Class Diagrams

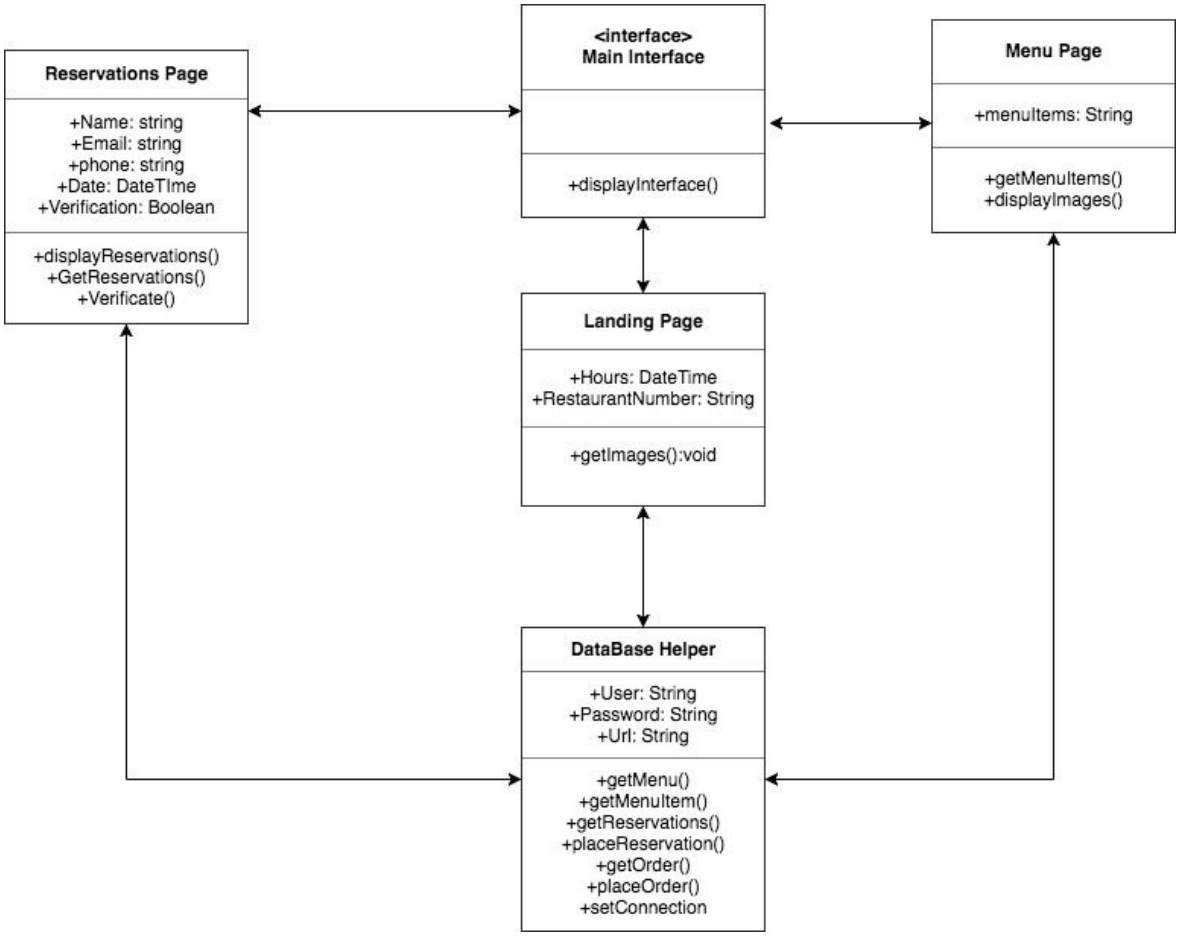
Desktop Application Class Diagram



Mobile Application Class Diagram



Website Class Diagram



Data Types and Operation Signatures

Desktop data types and operation signatures

Kitchen interface:

Kitchen Model Class	
Attributes	**Description**
- kitchenSection: int	To store respective kitchen section as part of a data structure of kitchen sections.
- Menu: List	To store/display menu list for cooks references.
- urlConection: socket	To establish bidirectional communication between kitchen and waiter applications.
- orderState: boolean	To show whether order is ready or not.

Kitchen Model Class	
Methods	**Description**
+ setKitchenSection(int):void	To store/update kitchen section.
+ setUrlConnection(int): void	To create the connection via designated port number.
+ setOrderState(boolean): void	To update order status.
+ getMenuList(): List	To display all menu items in the screen as requested.
+ getOrderState(): boolean	To check whether order is ready or not.

Admin Interface:

Admin Model Class	
Attributes	**Description**
- userName: String	Instance variable to store manager name.
- password: String	Instance variable to store a password.

Admin Model Class	
Methods	**Description**
+ setUserName(String): void	Setter function to store/update a new/existing name in data structure.
+ setPassword(String): void	Setter function to store/update a new/existing password in data structure.
+ setSchedule(List): void	Setter function to store/update a new/existing work schedule in data structure.
+ setReservation(object): void	Setter function to store/update a new/existing reservation in data structure.
+ getSchedule(): List	Getter function to display an existing work schedule.
+ getFloorStaff(): List	Getter function to display all employees working in restaurant floor.
+ getKitchenStaff(): List	Getter function to display all employees working in restaurant kitchen.
+ getReservations(): List	Getter function to display all existing reservations in the restaurant.
+ getReservations(String): String	Getter function to display an specific reservation.
+ getSells(): List	Getter function to display all existing sells.
+ getPrediction(): Obj	Getter function to get all prediction from our predicting model.

Floor Interface:

Floor Model Class	
Attributes	**Description**
- floorSection: int	Instance variable to store floor section number.
- urlConnection: Socket	Instance variable open a new connection between host and waiter.
- tableState: String	Instance variable to hold current table state(ready, dirty, occupied).
- tableNumber: int	Instance variable to store a table number.

Floor Model Class	
Methods	**Description**
+ setFloorSection(int): void	Setter function to store/update a new/existing floor section in data structure.
+ setTableNumber(int): void	Setter function to store/update a new/existing table number in data structure.
+ setConnection(String): void	Setter function to create a connection via designated port number.
+ setTableState(String): void	Setter function to update the current status of a table.
+ getFloorSection(): int	Getter function to display all existing floor sections and floor layout.
+ getTableNumber(): int	Getter function to display table number.
+ getTableState(): String	Getter function to display current table status.

Clock in/out Interface:

Clock Model Class	
Attributes	**Description**
- employeeName: String	Instance variable to store employee name.
- employeeID: int	Instance variable to store employee identification number.
- employeeBiometricPrint: Image	Instance variable to store employee fingerprint.

Clock Model Class	
Methods	**Description**
+ setEmployeeName(String): void	Setter function to store/update a new/existing employee name in data structure.
+ setEmployeeID(int): void	Setter function to store/update a new/existing employee ID number in data structure.
+ setEmployeeBiometricPrint(Image): void	Setter function to store/update a new/existing employee fingerprint in data structure.
+ getEmployeeName(): String	Getter function to display employee name.
+ getEmployeeID(): int	Getter function to display employee ID number.

DataBase Interface:

DataBase Class	
Attributes	**Description**
- url: String	Instance variable to store DBMS url.
- user: String	Instance variable to store DBMS user name.
- password: String	Instance variable to store DBMS password.

DataBase Class	
Methods	**Description**
+ setConnection(String url): void	Setter function to create connection between DBMS and desktop application.
+ setReservation(String...): void	Setter function to store/update a new/existing reservation in database(reservation table).
+ setMenu(List): void	Setter function to store/update a new/existing menu items in database(menu table).
+ setMenuItem(String): void	Setter function to store/update a specific menu item in database(menu table).
+ setSchedule(List): void	Setter function to store/update a new/existing work schedule in database(schedule table).
+ setDailySells(String): void	Setter function to store/update a new/existing sells in database(sells table).
+ getUserName(): String	Getter function to get specific employee name from DBMS for identification(login table).
+ getPassword(): String	Getter function to get specific employee password from DBMS for identification(login table).
+ getReservation(): String	Getter function to get specific reservation from DBMS (reservation table).
+ getReservations(): List	Getter function to get all existing reservations from DBMS (reservation table).
+ getMenu(): List	Getter function to get all existing menu items from DBMS (menu table).
+ getMenuItem(): String	Getter function to get a specific item in the menu from DBMS (menu table).
+ getSchedule(): List	Getter function to get all existing work schedules from DBMS (schedule table).
+ getDailySells(): List	Getter function to get all past sells from DBMS (sells table).

Mobile data types and operation signatures

LoginActivity:

LoginActivity Class	
Attributes	**Description**
- user: String	Instance variable for user
- password: String	Instance variable for user's password

LoginActivity Class	
Methods	**Description**
+ userLogin(): void	Function to check with DB to check if user is a valid user of system
+ onCreate(Bundle): void	Standard Android function that will create elements to be shown on screen
+ onClick(View): void	Event Handler function for when a button is pressed to call userLogin()

FloorsActivity:

FloorsActivity Class	
Attributes	**Description**
- FloorList: List	Android on screen list element for selecting floor
- TableList: List	Android on screen list element for selecting table
- Choicebtn: Button	Android on screen button element for triggering events

FloorsActivity Class	
Methods	**Description**
+ floorChoice(List): bool	Event handler function to set Floor when user chooses from list
+ tableChoice(List): bool	Event handler function to set Table when user chooses from list
+ onClick(View): void	Event handler function for when floor and table are selected to begin order

MenuActivity:

MenuActivity Class	
Attributes	**Description**
- menuList: List	Android on screen list element for selecting menu items
- quantity: int	Instance variable to increase quantity of selected menu items
- order: Object	Object that holds menu item and quantity
- finish: Button	Android on screen button element for triggering events

MenuActivity Class	
Methods	**Description**
+ onClickItem(View): order	Event handler function for when an item is clicked to create and order object and store it in there with chosen quantity
+ onClickerOrder(Order): void	Event handler function for when finish is pressed to send order object to OrderActivity class
+ post_order(data)	Function to populate order table of database using JSON object and hashmap
+ getmenu	Uses android volley function to get JSON data from database which is then parsed and used to populate the menuList
+ nextOrder()	Function called on press of order button which calls other functions, gets selected items and their quantities and calls post_order before displaying OrderActivity. Erroneous input on menu is handled in this function.

OrderActivity:

OrderActivity Class	
Attributes	**Description**
- currentOrder: List	Android on screen list element for passed order object to verify order before being sent
- orderedList: List	Android on screen list element for viewing sent orders to Kitchen
- sendBtn: Button	Android on screen button element for sending order to DB
- payBtn: Button	Android on screen button element for paying order; opens third-party screen

OrderActivity Class	
Methods	**Description**
+ onClick(View): void	Event handler function to view past sent orders

+ send(Order): void	Event handler function for order button to send order to DB for Kitchen
+ getAllOrders(): void	Function initially populates all orders for current user
+ getOrder(): void	Function populates items, quantities, and prices for current order
+ getPrices(): void	Function retrieves prices into array for current order's items
+ getSubtotal(): double	Function returns sum of items for current order
+ cookProgressSim(): void	Function shows progress of kitchen cooking current order
+ cookingFinishedNotif(): void	Function notifies waiter order is ready for pickup from kitchen

DBHelper:

DBHelper Class	
Attributes	**Description**
- user: String	Instance variable for user
- password: String	Instance variable for user's password
- Floor: Object	Instance object for restaurant floors
- Table: Object	Instance object for restaurant table
- menuItem: Object	Instance object for restaurant menu items
- order: Object	Instance object that holds menu items and quantities

DBHelper Class	
Methods	**Description**
+ userLogin(String...): bool	Check function for if given user and password vars and valid in the DB
+ getMenu(): menuItem	Getter function that returns menuItems from the DB
+ getFloor(): floor	Getter function that returns floors from the DB
+ getTable(): table	Getter function that returns tables from the DB
+ getOrder(): order	Getter function that returns orders from the DB
+ sendOrder(order): order	Function that sends order from DB to be sent to the Kitchen

Website data types and operation signatures

DBHelper Class	
Attributes	**Description**
- user: String	Instance variable for user
- password: String	Instance variable for user's password
- Url: String	Instance variable to store DBMS url.

DBHelper Class	
Methods	**Description**
+ setConnection()	Setter function to create connection between DBMS and desktop application.
+ placeReservations()	Function that places a new Reservation into the DB
+ getReservations()	Getter function that returns Reservations from the DB
+ getMenu()	Getter function that returns Menu from the DB
+ getMenuItem()	Getter function that returns specific Menu Item from the DB
+ sendOrder()	Function that sends order to DB to be sent to the Kitchen
+ getOrder()	Getter function that returns Order from the DB

Landing Page Class	
Attributes	**Description**
+ Hours: DateTime	The hours of operation that the restaurant is open

- RestaurantNumber: String	The restaurant's phone number

DBHelper Class	
Methods	**Description**
+ getImages()	Getter function that returns the images of the Items from the DB

Menu Page Class	
Attributes	**Description**
+ MenuItems:String	All the menu items available at the restauarnt

DBHelper Class	
Methods	**Description**
+ getMenuItems()	Getter function that returns the images of the Items from the DB
+ displayImages()	Displays the images of the items after getting them

Reservations Page Class	
Attributes	**Description**
+ Name: String	The name of the customer that wants to place a reservation
+ Email: String	The email of the customer that wants to place a reservation
+ Phone: String	The phone number the customer that wants to place a reservation
+ Date: DateTime	The time that the customer wants to place a reservation
+ Verification: Boolean	Whether the verification returns valid or not

Reservations Class	
Methods	**Description**
+ getReservations()	Getter function that returns the Reservations from the DB
+ displayReservations()	Displays the returned Reservations
+ Verificate	Verify the customer for the reservation

Traceability Matrix:

	Software Classes								
Domain Concepts	DatabseHelper	Kitchen	Admin	Floor	Clock	FloorsActivity	LoginActivity	MenuActivity	OrderActivity
Employee		*	*		*	*	*	*	*
Manager			*						
Host/Hostess			*	*		*			
Waiter/Waitress			*						*
Kitchen		*	*						*
Floor	*		*	*		*			
Table	*			*		*			
Menu	*	*						*	
tableStatus	*			*					
Reservation	*		*	*					

OrderItem		*							*
OrderQueue		*							*
Check									*
Clock					*				
ClockSchedule	*		*		*				
ClockIn					*				
Predictor			*						

Traceability Matrix: Concepts/Class relationship

- The DatabaseHelper is able to modify/retrieve the menu, reservations, schedules, table/floor status.

- Employee revolves around all the staff and as a whole has access to the entire system's activities.

- The kitchen class has control over all the food operations including orders and menu.

- Admin is accessed via the manager and can view all the staff information. It can also modify reservations and clocking.

- Floor intakes the table and reservation activity overseen by the host/hostess.

- Clock class encompasses all of the scheduling/clocking in and out.

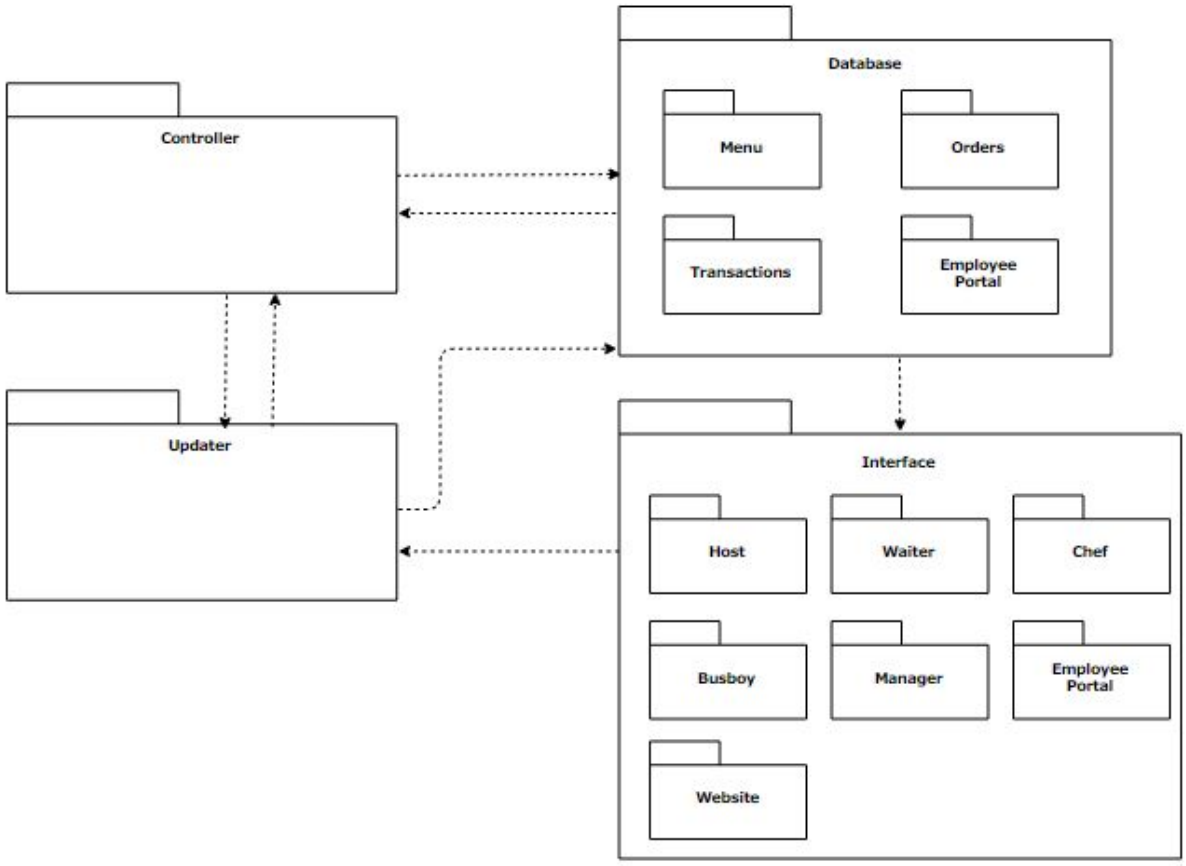
System Architecture and System Design

Architectural Styles:

The architecture style that we are using for our system is client server based system. The server stores data and executes functions which are initiated by the clients which would consist of the desktop and tablets. The server holds the database consisting of all the relevant information for the restaurant's operation such as order information, transaction information, employee information and table status. The functions to update,change or view this information is initiated by the client. The functions that the clients can access changes depending on the user permissions of who is associated with the client. The manager who is the highest ranking therefore having the most user permissions, has the most access to the server functions through the client. As an example, considering the manager as client will have permissions to do things that no other employee can such as changing the menu, viewing/changing the work schedule of an employee worked and checking transaction records. The manager therefore can make a request to the server to execute these function which will update the database and return the relevant information to the client. A client with less user permission such as a waiter will only be able to make limited server request related to his job such as requesting a change in table status and adding new orders. Since we will be using a cloud based server all our operations will be

dependent on a good network connection to allow quick communication between server and its clients for efficient restaurant operation. Delays in communication will result in delayed relay of information across the system from waiter to kitchen to manager which will be detrimental to the functioning of the restaurant. A cloud based system however has the advantage of not requiring to run,maintain and secure your own local server which will increase hardware investment costs.

Identifying Subsystems:



The interface subsystem will house everything all employee types (host, waiter, chef, busboy, manager) need to see, alongside the employee portal which will allow the the interface to talk to the updater. The interface subsystem will also have the website interface. The updater can then update any information in the database an employee would like to change through their devices. The database will display all necessary information to the interfaces of each employee, while housing the menu, orders, and transactions. All at the center of the system will be the controller subsystem, which will act as an intermediary between the updater and database.

Mapping Subsystems to Hardware:

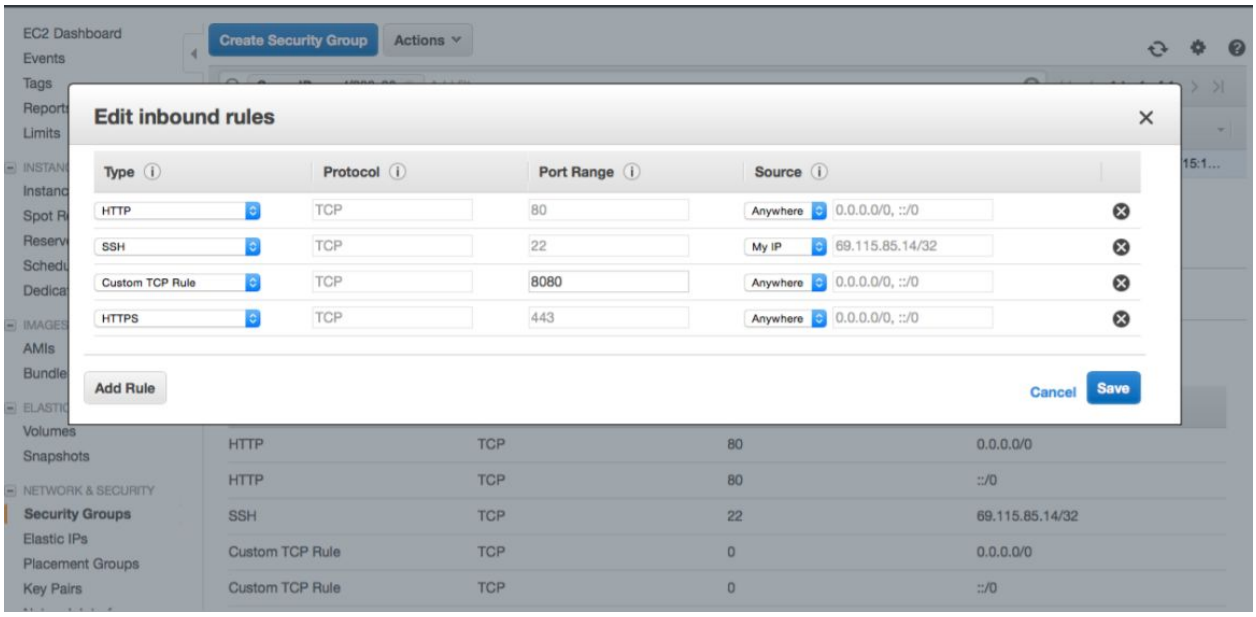
Because Restaurant Automation is a client-server architecture, multiple computers will be needed for the system. The server which has the database will be on a master computer, holding all the menu, order, transaction, and employee portal information from a remote location. A desktop app which will require another computer located at the restaurant will have the employee portal, manager interface, and host interface subsystems. The desktop app will also have all of the controller and updater subsystem functions. Tablets, which by definition are computers, will have the employee portal, water, chef, and busboy subsystems. The website subsystem can be accessed from any computer or mobile device to reserve tables or look at the menu.

Persistent Data Storage:

Our system requires data to be stored for an extended period of time for the purpose of record keeping and analysis by the manager to make adjustments to improve business as well as for the daily functioning of the restaurant. The orders, work schedules, clock in and out time, and menu are all stored in the database. The storage of the menu is necessary as the orders will be entered in the tablet and sent to the order section of the database based on what was present in the menu section of the database. One of the main benefits of using a cloud based database is we can have as much storage as we desire without having to worry about the space, cost and technical expertise required in installing,running and maintaining a local server to do all the tasks.

Network Protocol:

The system will be based on the Amazon Web Services Cloud for the desktop and web application. There will be a TCP/IP protocol that's configured with AWS for accessing the network. When SSHing onto the server (port 22), only the creator IP can access the online system to maximize security. There is a custom TCP Port 8080 that serves as the main source for uploading any JAR files through Tomcat's Apache as the applications will be Java, Javascript, and MySQL based.



Global Control Flow:

Execution Orderliness:

Restaurant Automation will execute in an event driven fashion initially, then in a linear fashion for the remainder of the procedures. The initial events would be if the customer is reserving a table for their party, they would input their party information and time through the website. Then whether or not the customer reserved, all customers would walk into the restaurant and tell the host their party's name and number of people in the party. If the customer did not reserve their table, the host would input the party's information, adding them to the queue.

The remainder of the global flow would be linear. Once the optimal table is ready for the customer's party, the party will be seated. After that, the server will input the customers' orders into the server's tablet, notifying the chefs of a new order. Once a chef takes the order and prepares it, he/she can notify the corresponding server when the order is ready. After the meal is sent to the customers, when the customers are ready to pay the bill, the server can update the tablet to let the system know the table is paying. Once they're done paying, the server can update the system that the table is dirty and ready for cleaning, letting the busboys know to clean the table. Finally, when the busboy is done cleaning the table, he/she can update the table in their tablet as a "ready to be seated" table.

Time Dependency:
Restaurant Automation uses clocks to keep track of how long an employee has worked during their shift between logging into the system and logging off. Other than that, the system is an event-response type since the control flow is dependent on if the user working on the current part of the flow is done and notifies the next part of the flow.

Concurrency:
Restaurant Automation will use multiple threads since there will be multiple systems going independently at the same time. One way multiple threads may spawn is if there are multiple customers reserving a table, waiting for a ready table, or putting in orders. This is controlled by putting the customers table or order requests in a chronological queue for the fairness of customers who acted first. Another instance of multiple threads is if the admin has to edit any information of a customer's order, menu, or transactions through the database. This won't create conflict with the other threads interacting with the customers as the thread is only interacting with database. There is no synchronization between threads as each thread handles different, independent actions.

Hardware Requirements:

The hardware components needed for the our system to operate are:

- A central desktop PC which will have the most functionality in using/changing the system via the desktop app with a monitor display
- Android tablets for the employees to carry out their duties via the mobile app.
- Customer web browsing device to access the restaurant website
- One router per floor for sufficient WiFi connection. Each terminal should have a minimum bandwidth of 1.5mbps download speed and 768 kbps upload speed.

Desktop

Hardware Component	**Minimum Requirements**
Processor	Dual Core CPU
RAM	4GB
HDD	128GB
Network Connection	Ethernet/Wifi card

Tablet

Hardware Component	Minimum Requirements
Processor	Quad Core CPU
RAM	2GB
HDD	16GB
Screen	Touchscreen display
Network Connection	Wifi card

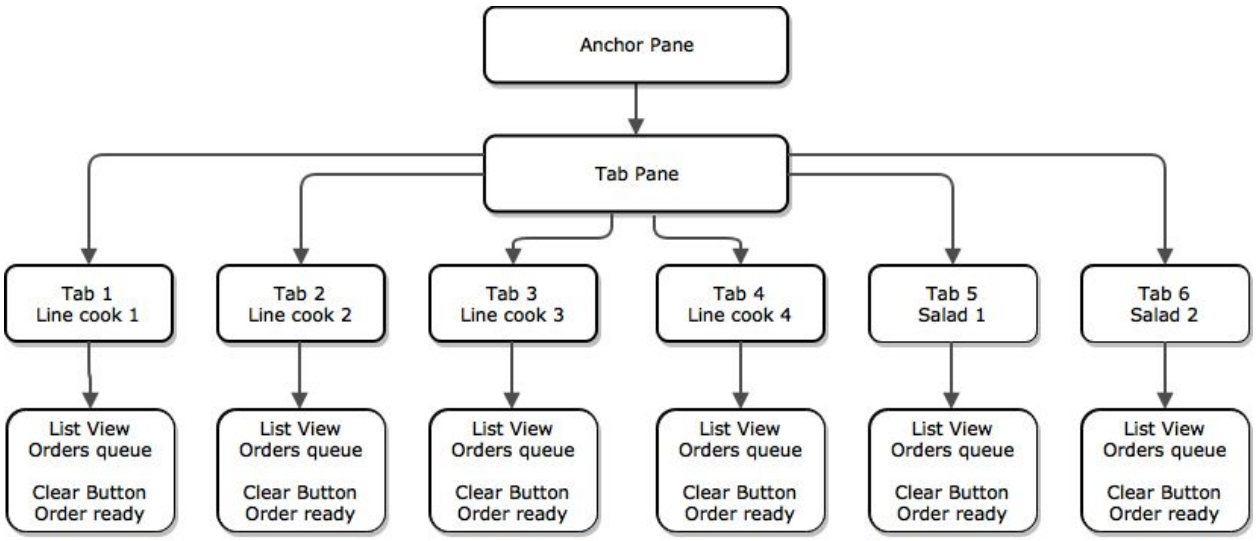
Customer Web Browsing Device

Any device that can access the internet using a web browser.

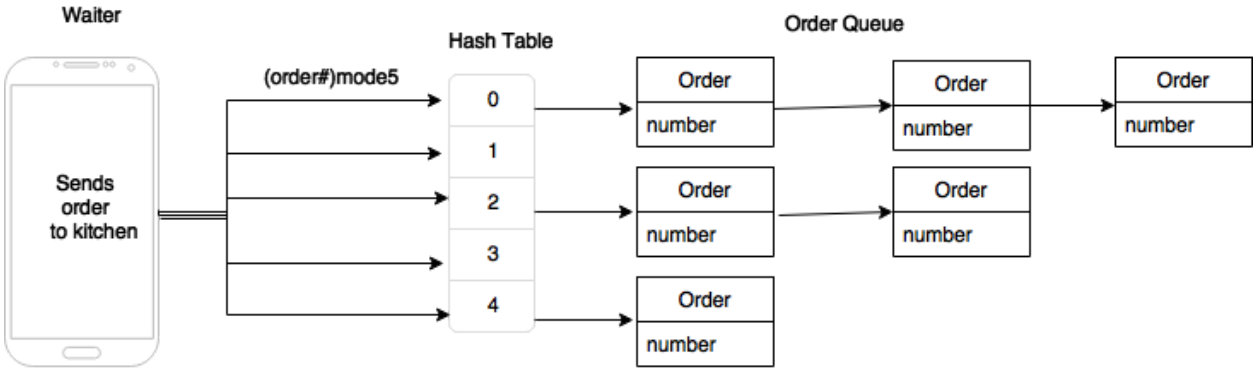
Algorithms and Data Structures

9.1 Kitchen Data Structure

UI data structure was implemented using XML tree structure architecture and the kitchen class model is constructed by a hash table priority queue and orders will be stored as they are sent in.



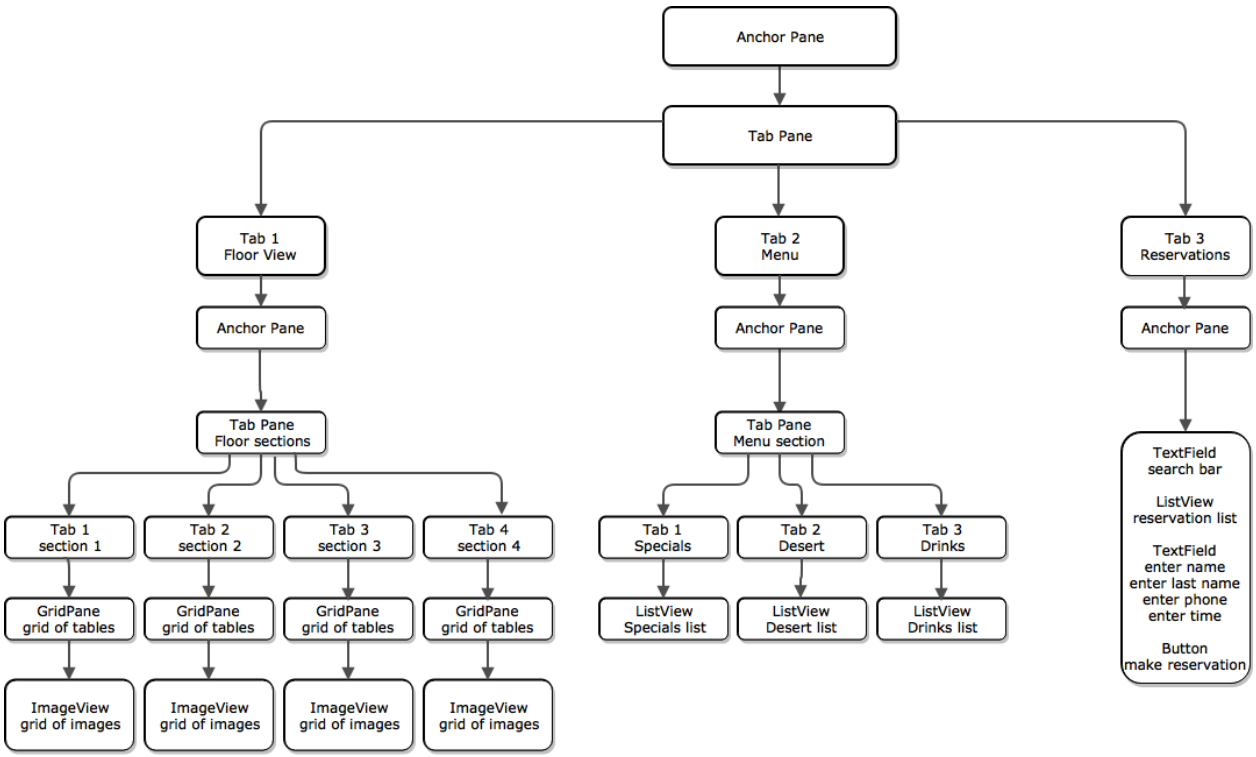
UI view data structure tree like diagram.



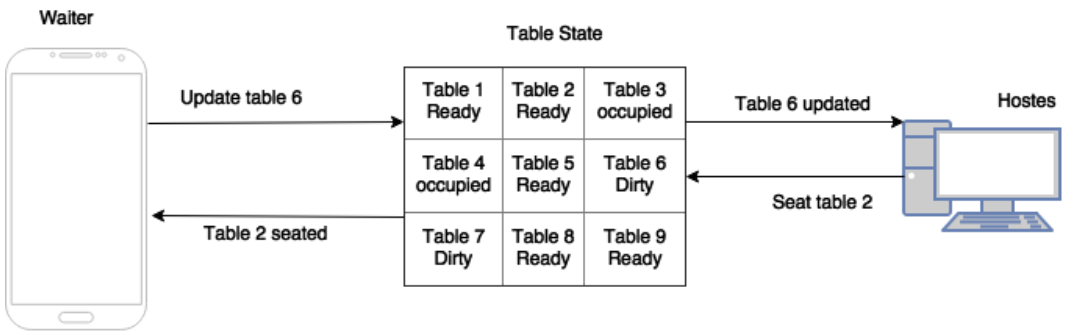
Kitchen model class data structure.

9.2 Floor Data Structure

UI data structure was implemented using XML tree structure architecture. All tabs contain different functionalities: tab 1 floor layout in order to see table's state, update table's state and to seat designated next in queue reservation. Client-server communication between floor layout and waiter's mobile app will be established via a port number using sockets and threads that will be constantly listen for any updates. Tab 2, menu, will work directly with our database system to retrieve menu items stored in DBMS and finally tab 3 also with direct communication with our DBMS for any requested reservation queries.



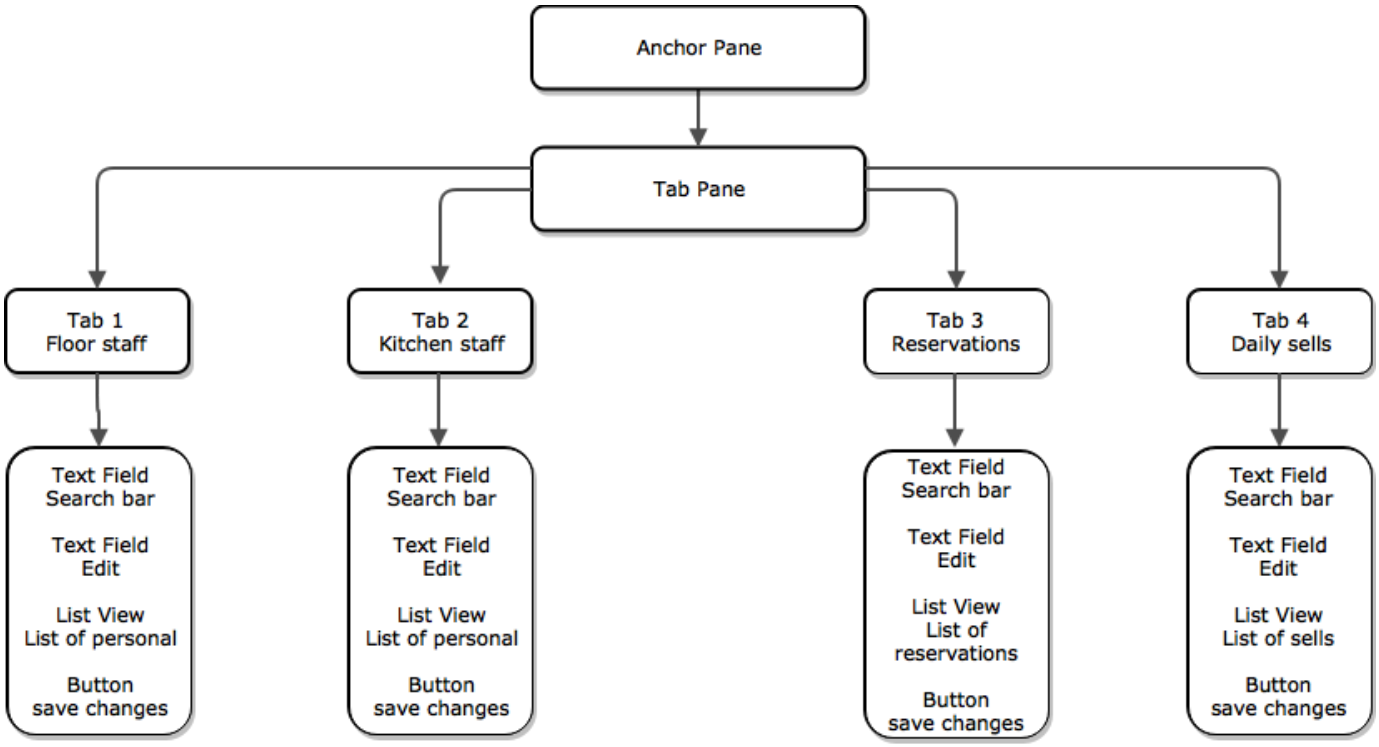
Floor UI tree structure.



Waiter-hostess communication.

9.3 Admin Data Structure

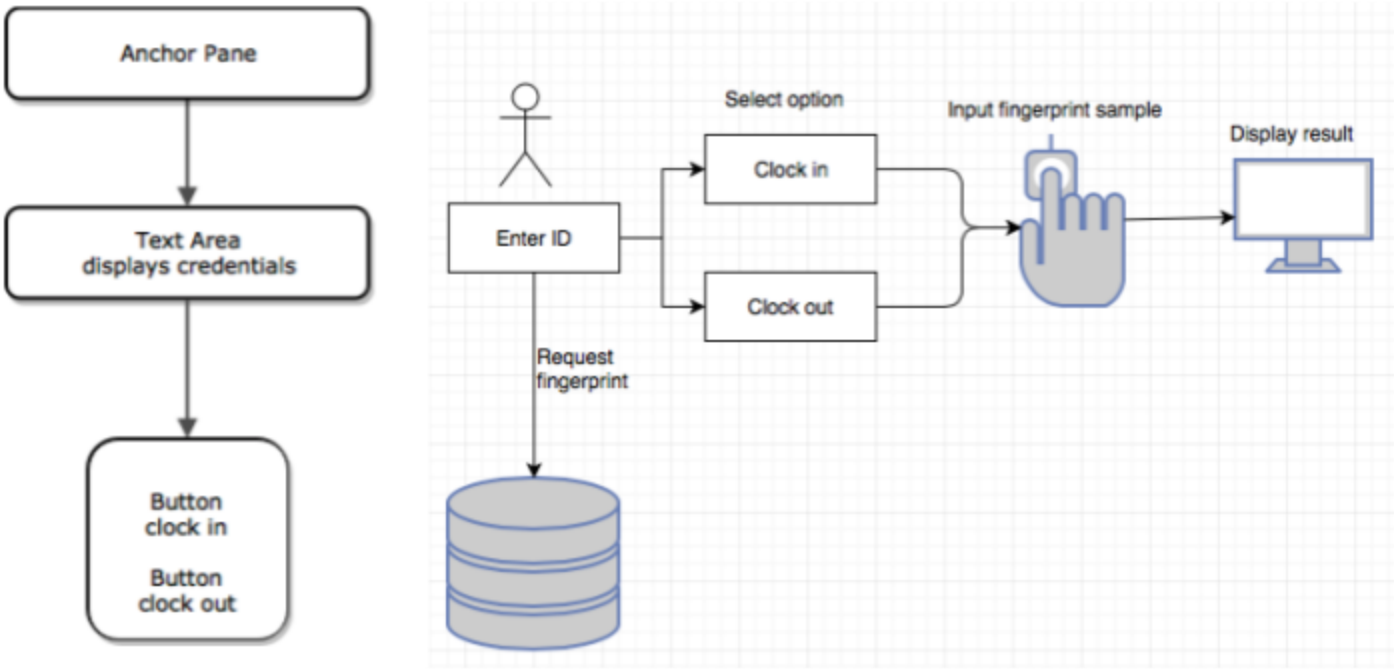
Admin UI also constructed in XML data structure format. Admin interface is intended for all management queries, i.e. chef needs to change the restaurant menu or manager needs to see an specific invoice. Admin works directly with our database management system. Its data structure will dependent on the database helper class using all the methods needed. DBMS will respond back to any admin query and the output will be displayed using ListView data structure.



UI view Data Structure tree like diagram.

9.4 Clock device Data Structure

Clock device UI also in XML structure will prompt user to place fingerprint for biometric recognition to then be compared with data stored in DBMS. Credential will be display as a ListView of objects. There will be no limit of entries for user to clock in. Time of transaction will be recorded and stored in DBMS as receipt. Same goes when user clicks out of work.



Data Structure tree like diagram.

9.5 Mobile app

9.5.1 Table Object Data Structure

The Table Object will be a custom object that holds different primitive types. The types are as follows: two int variables, one to represent the table number while the other represents the floor number, and three boolean values, one to show available status, another to unavailable status and lastly to show needs cleaning status. Only one boolean should be true at anytime since the table cannot be in several states simultaneously.

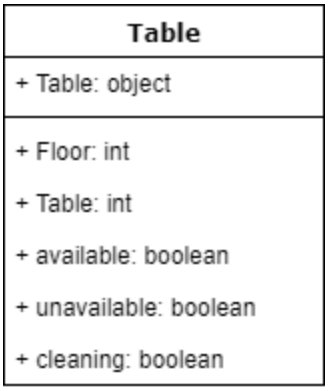


Diagram of Table Object

9.5.2 Order Object Data Structure

The Order Object is a Serializable Object that will pass through several different classes before being sent to the database. The Order itself will hold two int variables representing the table and floor on which the order is being placed for. It will also hold a Hash Map which will have a String representing the menu item and an int representing the quantity of the menu item.

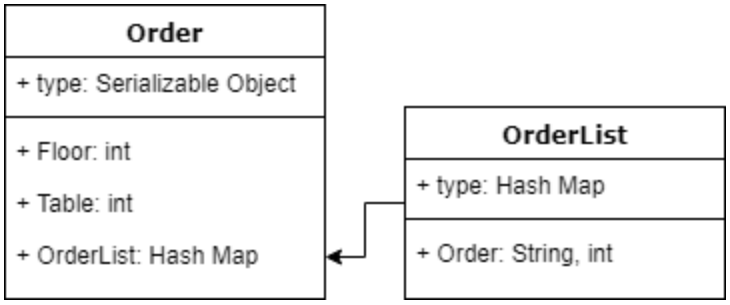


Diagram of Order object

9.5.3 Order Status List Data Structure

The Order Status List is a Hash Map that is retrieved from the database holding data on all ongoing sent orders for the current floor the user has selected. The Hash Map consists of orders that hold a String for order name (e.g. order 1, order 2,...,etc) and an int for progress (e.g. 0 for 0%, 50 for 50% done, 100 for complete). The Order Status List will be used to populate the Orders List View gui element so users can see the progress of the orders and when to retrieve them from the Kitchen.

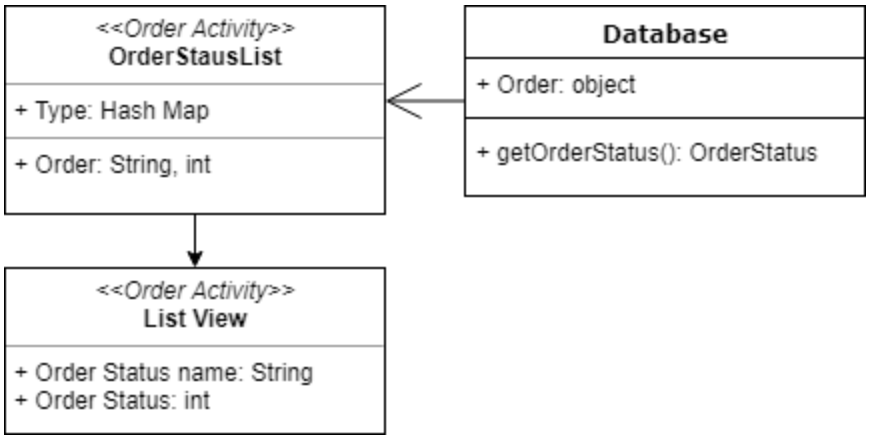
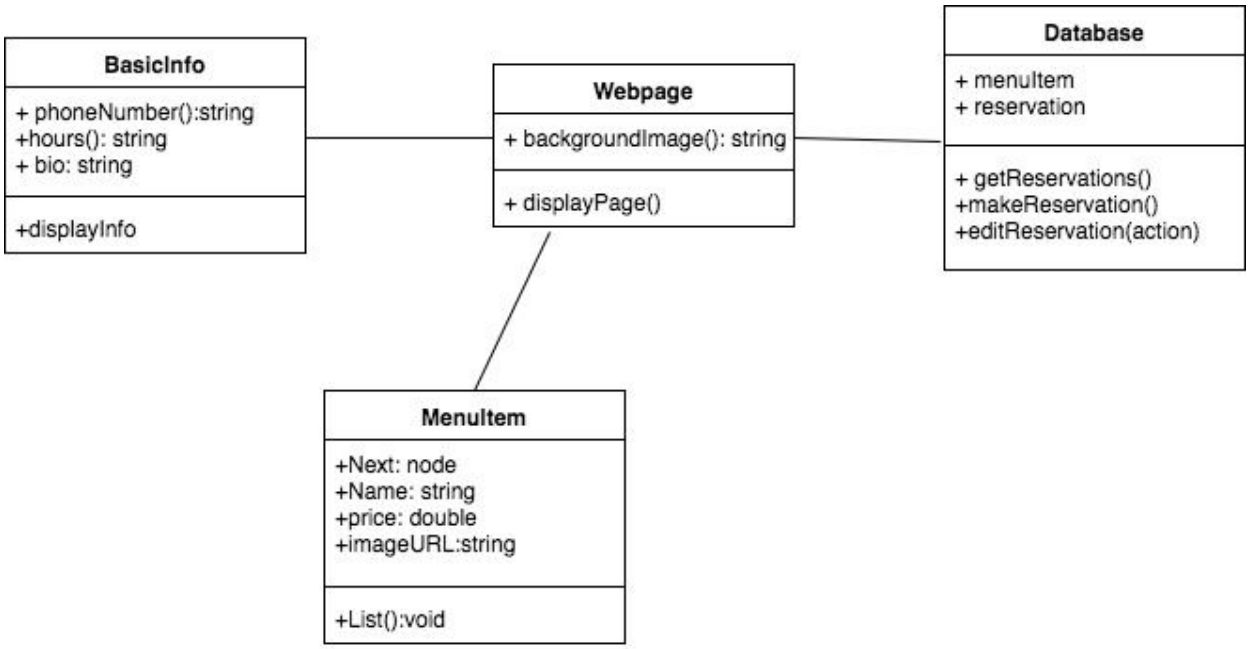


Diagram of how Order Status List is populated and used

9.6 Website

The basic information will be stored in an array. A linked-list will be used to hold the menu items because we want to list the items in order from appetizer to dessert but still be able to edit the list. We don't need random access to the menu items, they will all be displayed when the page is loaded.



User Interface and Implementation

The user interface for the tablets and desktop begin with a lock screen. Based on who logs in to the respective device the the functionality available to the user varys with the most functionality being given to the Manager. The tablets once logged into displays to the waiter the layout for the floor they are responsible for along with table status tables with customers waiting for their food and tables with customers whose orders need to be taken and empty tables where new customers can be seated. There is a menu button that shows all menu items on screen which can be selected to compile an order for a table. Once all the menu items are selected and the quantity of each is specified the place order button can be selected to go to the order confirmation screen. On the order confirmation screen there will be an option to go back to edit order, an option to place order which sends it to the kitchen and the total cost of the order. Once an order has been placed the table screen is presented with the status of the table changed from waiting to order to waiting for food. Once a customer has finished their meal and left the table the table status can be updated to ready for seating. The tablet will UI will be implemented in Android studio using buttons,image buttons,fragments and their functionality will be implemented using JAVA. In the beginning the UI will be very basic to implement functionality. Once functionality is finalised images and modification will be made to improve presentation

The website UI will have a straightforward layout showing information about the restaurant in the main page such as its location and menu. There will also be options to get contact information of the restaurant and make reservations. The reservation page will have a form structure requiring reserving customers name, the reservation time and the table size the customer wishes to reserve. The UI for the website will be implemented using HTML,CSS and JavaScript.

The Desktop App UI will also have a screen showing the tables. Both desktop and tablet floor screens are updated with the changes made by each other. The Desktop App if logged in by the manager, also has options to view/edit employee information, transaction records in addition to

everything the tablets can do. The Desktop App is also the only thing capable of making changes to the menu when the appropriate user is logged in. The desktop app UI implementation will made using JAVA and Scene Builder. The desktop app, tablets and website are all synchronised and updated to reflect the most recent information through communication via a cloud based database. The desktop app will also handle the clocking in of the employees by mean of a fingerprint scanner to ensure accurate representation of employee attendance and hours worked. It will also have a prediction feature that can be used to determine how busy the day will be based on the restaurant's history with busy days.

Design of Tests

Website:

Test Case Identifier: TC-W1 **Function Tested:** onClickRedirect(): void **Use Case Tested:** UC - 6 LoginActivity, UC-3 FloorsActivity, UC-1 OrderActivity **Pass/Fail Criteria:** Test will pass if directed to clicked page.	
Test Procedure:	**Expected Results:**
Step 1. Call Function - Pass	Desired view page is brought on the screen.
Step 2. Call Function - Fail	Function will print page does not exist or cannot be accessed and exit(-1).

Test Case Identifier: TC-W2 **Function Tested:** makeReservation(): void **Use Case Tested:** UC - 4 Reservation **Pass/Fail Criteria:** Test will pass if reservation is attempted with correct verification and the reservation calendar is updated with the new reservation	
Test Procedure:	**Expected Results:**
Step 1. Call Function - Pass	Reservation is made in reservation calendar
Step 2. Call Function - Fail	Reservation is not made and the calendar doesn't update

Test Case Identifier: TC-W3 **Function Tested:** makeReservation(): void **Use Case Tested:** UC - 4 Reservation **Pass/Fail Criteria:** Test will pass if reservation is attempted with incorrect verification and the reservation calendar doesn't have the attempted reservation	
Test Procedure:	**Expected Results:**
Step 1. Call Function - Pass	Reservation is not made in reservation calendar and the user is told the verification is incorrect/not verified yet.
Step 2. Call Function - Fail	Reservation is made and the calendar updates to show it

Desktop:

Test Case Identifier: TC-01 **Function Tested:** onClick(View): void **Use Case Tested:** UC - 6 LoginActivity, UC-3 FloorsActivity, UC-1 OrderActivity **Pass/Fail Criteria:** Test will pass if directed to clicked page.	
Test Procedure:	**Expected Results:**
Step 1. Call Function - Pass	Desired view page is brought on the screen.
Step 2. Call Function Fail	Function will print page does not exist or cannot be accessed and exit(-1).

Test Case Identifier: TC-02 **Function Tested:** placeReservations() **Use Case Tested:** UC-4 Reservation **Pass/Fail Criteria:** Test will pass if reservation is placed in the DB	
Test Procedure:	**Expected Results:**
Step 1. Call Function - Pass	New reservation is added onto the reservation calendar.
Step 2. Call Function Fail	Function will return NULL if reservation is failed to be added

Test Case Identifier: TC-03 **Function Tested:** getMenu(): menuItem **Use Case Tested:** UC-1 Order, UC-2, Serve **Pass/Fail Criteria:** Test will pass if menu is retrieved from the database.	
Test Procedure:	**Expected Results:**
Step 1. Call Function - Pass	Menu option appears on screen
Step 2. Call Function Fail	Returns null pointers if Menu is failed to be retrieved

Test Case Identifier: TC-04 **Function Tested:** getFloor(): floor **Use Case Tested:** UC-3 Seat, UC-4 Reservation **Pass/Fail Criteria:** The function passes when the Floor page is displayed upon request	
Test Procedure:	**Expected Results:**
Step 1. Call Function - Pass	The table layout of the floors are shown correctly when retrieved
Step 2. Call Function Fail	Function prints failed to open floor layout and exit(-1)

Test Case Identifier: TC-05 **Function Tested:** getTable(): table **Use Case Tested:** UC-3 Seat **Pass/Fail Criteria:** The function passes when table availability is shown	
Test Procedure:	**Expected Results:**
Step 1. Call Function - Pass	The desired table along with the availability colors are shown correctly
Step 2. Call Function Fail	Function returns NULL pointer upon request

Test Case Identifier: TC-06 **Function Tested:** getOrder(): order **Use Case Tested:** UC-1 Order **Pass/Fail Criteria:** The function passes when the order is retrieved	
Test Procedure:	**Expected Results:**
Step 1. Call Function - Pass	The order is sent to the DB retrieved by the kitchen
Step 2. Call Function Fail	Function returns NULL pointer upon request

Test Case Identifier: TC-07 **Function Tested:** getPrediction(): data **Use Case Tested:** UC-6 Admin **Pass/Fail Criteria:** The function passes when admin requests query.	
Test Procedure:	**Expected Results:**
Step 1. Call Function - Pass	All information is displayed in the UI
Step 2. Call Function Fail	If there is data was not processed or an exception/interrupt occurs.

Mobile Application:

Test Case Identifier: TC-M1 **Function Tested:** userLogin(String, String): bool **Pass/Fail Criteria:** Test will pass if user logs in and loads user interface.	
Test Procedure:	**Expected Results:**
Step 1. Call Function - Pass	If the username and password strings match the system returns true and allows the user interface to load for the user. If the username and password do not match, the system returns false and does not load user interface.
Step 2. Call Function - Fail	If the username and password do not match, the system returns false and does not load user interface.

Test Case Identifier: TC-M2 **Function Tested:** onClick(View): void **Pass/Fail Criteria:** The test will pass if the view is updated with the correct interface upon clicking.	
Test Procedure:	**Expected Results:**
Step 1. Call Function - Pass	The view updates to the correct interface when the corresponding interface object is clicked.
Step 2. Call Function - Fail	The view can’t update the corresponding interface, and the user interface stays the same and doesn’t update.

Test Case Identifier: TC-M3 **Function Tested:** post_order(hashmap): void **Pass/Fail Criteria:** The test will pass if the order is passed on to the database and added to the order queue.	
Test Procedure:	**Expected Results:**
Step 1. Call Function - Pass	Order is added to the order queue in the database.
Step 2. Call Function - Fail	The order will be unable to be added to the order queue, and an error message will be displayed.

Test Case Identifier: TC-M4 **Function Tested:** onClickItem(View): order **Pass/Fail Criteria:** Test will pass if the view updates to show item details, and customization options for the chosen item.	
Test Procedure:	**Expected Results:**
Step 1. Call Function - Pass	The view updates and item details and customization options are shown for the current item. System knows what the current item is that is about to be ordered and stores it in a variable.
Step 2. Call Function - Fail	The item view can’t be loaded and an error message will be displayed. The user interface will not update.

Test Case Identifier: TC-M5 **Function Tested:** onClickOrder(View): void **Pass/Fail Criteria:** Test will pass if the view updates to show all items ordered with price details and potential customization details.	
Test Procedure:	**Expected Results:**
Step 1. Call Function - Pass	The view updates to show all items ordered with price details and potential customization details. System knows what the current order is that is about to be ordered and stores it in a variable.
Step 2. Call Function - Fail	The order view can't be loaded and an error message will be displayed. The user interface will not update.

Test Case Identifier: TC-M6 **Function Tested:** floorChoice(Object floor): bool **Pass/Fail Criteria:** Test will pass if the view updates to show the floor details associated with the requested floor.	
Test Procedure:	**Expected Results:**
Step 1. Call Function - Pass	The view updates to show the table details associated with the requested floor. System knows what the current floor requested is by a variable associated with the current user.
Step 2. Call Function - Fail	The floor view can't be loaded and an error message will be displayed. The user interface will not update.

Test Case Identifier: TC-M7 **Function Tested:** orderNext(): void **Pass/Fail Criteria:** Test will check if no items are selected or if an item is selected and no quantity for that item is selected	
Test Procedure:	**Expected Results:**

Step 1. Call Function - Pass	Order is sent to database and page containing pending orders is displayed
Step 2. Call Function - Fail	Error messages displayed based on if no item were checked or if item was checked but quantity was not specified for that item

Test Case Identifier: TC-M8 **'Function Tested:** tableChoice(Object table): bool **Pass/Fail Criteria:** Test will pass if the view updates to show the table details associated with the requested table.	
Test Procedure:	**Expected Results:**
Step 1. Call Function - Pass	The view updates to show the table details associated with the requested table. System knows what the current table requested is by a variable associated with the current user.
Step 2. Call Function - Fail	The table view can't be loaded and an error message will be displayed. The user interface will not update.

History of Work, Current Status, and Future Work

Desktop :

- By Report 1 the following has been completed:
    - Kitchen, Admin, Clock system, Desktop system UIs implemented.
- By Report 2 the following has been completed:
    - All desktop system functionality implemented
    - TCP-IP network connection between mobile and desktop established
    - DBMS and relational database using MySql implemented
    - Biometric fingerprint recognition research
- By Report 3(current status and future work) the follow has been/needs to be implemented:
    - 80% Biometric fingerprint recognition implemented in clock system.
    - Search for relevant restaurant data to build a predictive model.
    - Kitchen order queue

Mobile:

- By Report 1 the following has been completed:
    - Basic Framework of mobile application completed (pages and moving between pages)
- By Report 2 the following has been completed:
    - Login functionality of users
    - Socket connections to main application for networking
    - More UI elements implemented (lists for menus and orders)
    - Populating Menu list with Menu items.
    - Populating Order page with order data.
- By Report 3(current status) the follow has been implemented:
    - Payment Processing through Braintree.
    - Enable acquiring of updated menu from database.
    - Implemented addition of orders to database

- Implemented order status bar.

Website:

- By Report 1 the following has been completed:
  - Basic Framework of website completed (tabs and outline of pages)
- By Report 2 the following has been completed:
  - Adjustment of CSS of website layout
  - Addition of more functionalities (team profiles, reservation requests)
  - Addition of project general page and description
- By Report 3 the following will be implemented:
  - Navigation from reservation request to database
  - Full implementation of every team member's profile
  - Full menu list and completion of site
- Mobile

  Working on this project, we had to evolve certain aspects of our application. The application's evolutions include redesigning the UI of the Menu and Order pages to accommodate new features. For Menu pages the functionality was changed from just setting up the order of customers to including sending it to the database instead of a final check in the order page. In the order page, it was changed from having a list of checking the current order before being sent to the database to removing that and having two lists, one to check the sent orders from the user and the other to view the items ordered in a selected order. A new button to handle payment process was added to allow customers to pay for their meals at the table immediately.

- Website

  Use cases handled by the website is mainly just UC-4 for reservations. The rest of the website is mostly static information about the restaurant. Currently, the website has been made with the image and information. The functionality of reservations hasn't been implemented yet but that will be the main focus from now on. This will be complete by the second demo. We also plan on implementing a feature that shows each group member's profile and contribution to the project.

Breakdown Of Responsibilities

- Desktop Responsibilities: subdivided into 4 responsibilities and each will be tested independently

  Desktop interface "Floor" (**Leonardo Roman**):
  - mobile-desktop communication using sockets and threads
  - Reservation queries.
  - Menu

  Clock device (**Leonardo Roman**):

- Biometric recognition
- Check credential in DBMS

Admin (**Leonardo Roman**)
- DBMS queries
- Predictive Model

Kitchen (**Eric**):
- Order queue

- Mobile Responsibilities
  Each member will handle testing of their modules/classes. Integration will be handled by all members (Jan, Sadiq, Christian) equally.

  Login Module - Jan Matthew Miranda
  Database Helper Class - Jan Matthew Miranda
  Floor Activity Class - Sadiq Rehan
  Menu Activity Class - Sadiq Rehan/Christian Remolado
  Order Activity Class - Christian Remolado

- Website Responsibilities
  Landing home page - Peter
  Menu - Peter
  Reservation function - Peter & Kevin
  Profile functionality - Kevin
  Template of Website - Kevin

References

1. **Group 3's Report 2015 in formatting of document:**
    a. http://www.ece.rutgers.edu/~marsic/books/SE/projects/Restaurant/2015-g3-report3.pdf
2. **Group 4's Report 2014 in formatting of document:**
    a. http://www.ece.rutgers.edu/~marsic/books/SE/projects/Restaurant/2014-g4-report3.pdf
3. **UML Diagrams designed in:**
    a. http://www.draw.io
    b. https://www.lucidchart.com
4. **User Story Points:**
    a. https://www.atlassian.com/agile/project-management/estimation
    b. The website was used to determine how many story points should be awarded for each problem
5. **User Effort Estimation:**
    a. https://www.tutorialspoint.com/estimation_techniques/estimation_techniques_use_case_points.htm
    b. Website was used to determine how to properly calculate user estimations for technical and environmental factors.
6. **Gantt Charts:**
    a. http://www.gantt.com/
    b. The website was used to figure how to create and a Gantt chart and how to use the tool effectively
7. **UI Mobile Application Mockup**
    a. https://www.fluidui.com

8. Hardware Requirements

https://support.revelsystems.com/hc/en-us/articles/204251049-What-are-the-bandwidth-requirements-
Determining the bandwidth and router usage for an internet dependent restaurant.

Effort Breakdown

	Jan	Kevin	Eric	Peter	Christian	Leonardo	Sadiq
Problem Statement	40%		30%		15%	15%	
Glossary			60%	40%			
Functional Requirements	20%	30%				30%	20%
Nonfunctional Requirements		15%				85%	
UI Requirements	25%			25%	25%		25%
StakeHolders & Actors	30%		30%	20%			20%
Use Case Casual	100%						
Use Cases Full Description		33.3%		33.3%		33.33%	
System Sequence Diagram		25%			50%		25%
Preliminary Design	**60%**			**25%**		**15%**	
Effort Estimation	**20%**		**60%**		**20%**		
Concept Definitions		**20%**	**40%**		**20%**		**20%**
Association Definitions			**20%**				**80%**
Attribute Definitions			**20%**	**15%**	**65%**		
Traceability Matrix						100%	
Domain Model						**100%**	
Traceability Matrix							**100%**
System Operation Contracts	**100%**						

Plan of Work	**60%**		**40%**				

Effort Breakdown Cont. Report 2

	Jan	Kevin	Eric	Peter	Christian	Leonardo	Sadiq
Interaction Diagrams				25%	25%	25%	25%
Class Diagram	**40%**			**20%**		**40%**	
Data Types and Operation Signatures	**35%**			**30%**		**35%**	
Traceability Matrix			**100%**				
System Architecture and System Design			33.33%		33.33%		33.33%
References			33.33%		66.67%		
Alg's & data struct	40%			20%		40%	
UI Design and Implementation							100%
Testing Design			60%	20%	20%		
Doc Merge	**33.33%**		**33.33%**			**33.33%**	
Proj. Coord/Progress	60%					40%	
Plan of Work	**35%**			**30%**		**35%**	
Total	258.33		259.99	145	144.99	233.33	158.33
Percentage out of 1,200	21.53%	0	21.66%	12.08 %	12.08%	19.45%	13.19%