****

**CentRes**

**Software Requirements Specification**

**Version 2.1**

**Team Number: 10**

**Project Manager: Katherine Landsman**

**Mentor: Susan Rizzo**

**Team Members: Carson Pribble, Chase Jamison, David Utshudiema, Mohamed Ibensilalen, Sergio Mainville**

**Revisions**

| **Version** | **Primary**  **Author(s)** | **Description of Version** | **Date Completed** |
| --- | --- | --- | --- |
| V.1.0 | Carson, Chase, David, Mohamed, Sergio, Katherine | First draft of Software Requirements V1 | 10/13/22 |
| V.1.1 | Carson, Chase, David, Mohamed, Sergio, Katherine | Finalized V1 of Software Requirements | 10/20/22 |
| V.2.0 | Carson, Chase, David, Mohamed, Sergio, Katherine | First draft of Software Requirements V2 | 10/30/22 |
| V.2.1 | Carson, Chase, David, Mohamed, Sergio, Katherine | Finalized draft of Software Requirements V2 | 11/4/22 |

**Review History**

| **Reviewer** | **Version Reviewed** | **Date** |
| --- | --- | --- |
| Susan Rizzo | V.1.0 | 10/19/22 |
| Susan Rizzo | V.2.0 | 10/31/22 |

**Table of Contents (Version 2.1)**

1. Introduction
   1. Project Objectives
   2. Project Scope
   3. Project Overview
2. General Description
   1. Project Features / Functions
   2. User Stories
   3. Use Cases
3. Team Collaboration and Documentation Tools
4. Project Management Plan
5. Requirement Specifications
   1. Business Requirements
   2. User Requirements
   3. Functional Requirements
   4. Non-Functional Requirements
   5. Implementation Requirements
6. High-Level Design
   1. Security
   2. Hardware
   3. User Interface
   4. Architecture
   5. Database
   6. Top-Level Classes
   7. Data Flow and States
   8. Reports
7. **Introduction**
   1. Project Objectives:

Allow for employees to easily interact with customers’ orders via an app and reduce miscommunication. Provide centralization of all server tables allowing servers to better manage and prioritize orders. App will help cut down on customer wait time by transmitting information directly to the kitchen.

* 1. Project Scope:
* App will not automatically rotate servers, the host will manually achieve this
* App will provide order capability
* App will provide order management including creating, deleting, adding, and removing items from an order.
* App will provide the host and manager the ability to manage servers.
* Host will have upgraded server privileges with the app to assign servers to tables
* Managers will have upgraded admin privileges within the app to edit customer orders
* Menu items will be stored in a database which only the manager will have the ability to access
* App will keep track of volume of items sold each day for tracking and reporting purposes
* App will not have timesheet functionality. (Keeping track of hours worked)
* App will not keep track of employee’s weekly schedules.
* App will not have the functionality for communication between users (servers, managers, etc.)
  1. Project Overview:

Servers will have an easy-to-read display that keeps track of things such as menu items, prices, order time, and order status. They can also view their current tables and see the current bill/tab for each table, see the average order time, and display information associated with a table. Line cooks/Chefs will see the current orders placed and the time they were placed. They can notify the server when an order is ready. The application will display the menu and allow the manager the ability to add or edit items in an order. When viewing a table, the server/manager can view or distribute an itemized bill to the customer. The application will display all tables allowing host/hostess to assign tables to specific servers.

1. **General Description**
   1. Project Features / Functions
      * Server management (allows host to assign tables)
      * Table management (track customer orders, and generate itemized bill)
      * Order management (server/kitchen order tracking)
      * Menu management (CRUD operations)
   2. User Stories
      * As a server, I want to only see available tables so I can reserve it for the customer.
      * As a server, I want to place an order in my device and send it directly to the kitchen.
      * As a line cook, I want to receive notification for every new order and can mark it ready after preparation.
      * As a server, I want to receive notification from the kitchen on every order that is ready so I can mark it completed after serving it to the right table.
   3. Use Case
      * **Name:** Server

**Actor:** Server

**Basic Use Case Description:**

* Server reserves available table for the customer.
* Server sends order to the kitchen.
* Server gets notification of order completion and marks the order complete. after order is served to the customer.
  + - **Name:** Line Cook

**Actor:** Line Cook

**Basic Use Case Description:**

* Line cooks receive placed order with time stamps.
* Line cooks check off individual items within order.
* Line cooks mark the order ready after all items within the order are prepared.
  + - **Name**: Host

**Actor:** Host

Basic Use Case Description:

* Host can manage table availability.
* Host can assign tables to servers.
  + - **Name:** Manager

**Actor:** Manager

**Basic Use Case Description:**

* Manager can edit orders.
* Manager can alter bill.
* Manager can view overall wait times for all orders.
* Manager can view time stamps on orders.

1. **Team Collaboration and Documentation Tools**

* GitHub
  + Used as a VSC
* GitHub Projects
  + Tracking project tasks in a centralized location
* Microsoft Teams
  + Used for project communication

1. **Project Management Plan**

* Agile Principles
  + Allows for project transparency, and task optimization
* Scrum Methodology
  + Encourages higher productivity

1. **Requirement Specification**
   1. **Business Requirements**

| Requirement ID | Requirement Description | MOSCOW |
| --- | --- | --- |
| BR1 | System must be available on both IOS and Android | M |
| BR2 | Effectively manage tables and orders | M |
| BR3 | Weekly efficiency reports are generated | M |
| BR4 | Keep long term data in a cloud or off-site server database with a data storage model allowing for frequent input of data but infrequent querying of that data (AWS) | S |
| BR5 | Rolling out special opportunities and events for customers in the rewards program | S |
| BR6 | Provide real-time business metrics to track sales, operations, and customer service performance. | M |

* 1. **User Requirements**

| Requirement ID | Requirement Description | MOSCOW |
| --- | --- | --- |
| UR1 | Host staff can manually mark tables as: a) Open for seating. b) Seated. c) Need bussing. | S |
| UR2 | Manager can edit orders, alter bill, overview of wait times, view time stamps in order | M |
| UR3 | Chef can view orders, time stamps for orders, and mark as ready | M |
| UR4 | Waiter can view orders, view tables, time stamp for orders, alter bill, view menu | M |
| UR5 | Waiter can add to order, print check, | M |
| UR6 | User can login with username and password | M |

* 1. **Functional Requirements**

| Requirement ID | Requirement Description | MOSCOW |
| --- | --- | --- |
| FR1 | Customer rewards program | C |
| FR2 | Display menu when requested | M |
| FR3 | Display orders, bill, wait times, and time stamps when requested | M |
| FR4 | Allow user to login when successfully entering in user and password | M |
| FR5 | Use 2FA for manager login to secure admin privileges (two factor authentication) | S |
| FR6 | Keep track of volume over time to create a predictive model for future use | S |
| FR7 | Require user log back in after tablet/device is idle for more than 60 seconds | S |

**5.4 Non-Functional Requirements**

| Requirement ID | Requirement Description | MOSCOW |
| --- | --- | --- |
| NFR1 | Database should update in a timely manner to keep communication between users accurate | M |
| NFR2 | Password must meet defined password criteria | M |
| NFR3 | Must not allow false input to a field | M |
| NFR4 | Users should only be able to access what they have permission to see | M |
| NFR5 | Can scale the restaurant table display volume and placement based on remodeling and rearrangement | S |
| NFR6 | Use of local database for efficiency and external/cloud-based database for long term data collection for aggregation | S |
| NFR7 | Use a third-party security/authentication system to avoid unwanted manipulation | C |
| NFR8 | Keep track of each table as an object with data attributes such as item orders per seat, bill cost, wait time | S |
| NFR9 | Implement customer rewards program with customer email | C |
| NFR10 | The system should support enough logged in users at any given time to satisfy the restaurants requirements for access frequency | M |

* 1. **Implementation (Performance) Requirements (Optional)**

| Requirement ID | Requirement Description | MOSCOW |
| --- | --- | --- |
| IR1 | The app should not go down due to an increase in users. | M |
| IR2 | The application should always load in less than 0.03 seconds. | C |

1. **High-level Design**
   1. **Security** 
      * Data
        1. Data in rest
           1. Ensure that the cloud service in use encrypts stored data so it is not easily attainable.
        2. Data in use
           1. Control access to viewable data in use by using least privileged access determined by user’s roles.
        3. Data in transit
           1. Usage of a cloud VPN/VPN as a service
   2. **Hardware** 
      * Network
        1. Wireless router
        2. Switch
        3. Cabling
        4. Cat6
        5. Universal power supply (UPS)
      * Tablet
        1. Application will be able to be used on IOS, Android, or any tablet OS that has web browser capability.
      * Printer
        1. Receipt printing for customer bills
        2. Server shift report
        3. Manager daily monetary report
   3. **User Interface**
      * Specifications

6.3.1.1 Each user type will have their own interface that will automatically refresh periodically and allow for manual page refreshing.

6.3.1.2 Once the URL is opened, a user will need to enter their unique ID/Username as well as password to access their interface.

6.3.1.3 The unique versions of the UI

6.3.1.3.1 Manager UI

This will have a top-down layout of the tables, each numbered and color-coded based on status (i.e., Seated, Unseated, Needs Bussing). There will also be an element in the form of a dynamic vertical scrolling window that shows each server on that shift that is clocked in. Each entry with a server’s name will be expandable to show their section (i.e., The table numbers they are assigned). The tables in their section will be expandable to show orders that have been pushed to the back-of-house team. The manager UI should also contain a field to query item availability (i.e., Menu Items, not raw ingredients). The manager’s account can comp items/modify or apply promos for a list of reasons when accessing any table in the restaurant. The manager UI will also include the ability to send predetermined pings (messages that are pre-determined) to servers on their server list. Those pings will be sent to the server that was chosen. Common pings could include things like a table needing to be bussed, notification that they have been seated (guests sitting down at a table in their section, the hosting staff will have this ability as well) or that they are needed at the line for clarification on an order.

6.3.1.3.2 Line/Back-Of-House UI

This will show incoming orders in the order that they were received. The interface (preferably touch screen monitor but mouse access works as well) will be accessible to any back-of-house staff and they will be able to send a notification to the server that submitted the order that it has been completed and is ready for pick-up. This should keep the volume of food sitting under hot-lamps or on a tray line moving quickly. The back-of-house UI will not be user based but instead accessible by any workers on the line. This interface will require a manager PIN to be opened, and once opened, it stays open for use by all line workers. Individual user accounts should not be necessary for back-of-house staff.

6.3.1.3.3 Server UI

This will primarily display a window for their currently open tables. Hosting staff will seat guests at tables based on their server section rotation and send a ping to the server in that section that will open a new table object in that server’s main UI window. They can click the table as a button to access the menu. The server will be able to increment seats and enter the order for each seat. Upon clicking a seat in a specific table’s interface, the server will then have access to the menu which will be split into sections (i.e., Beverages, Appetizers, Entrées, etc.). Once each order is entered, there will be a button that will send the order to the back-of-house’s queue of orders. Each table can be accessed multiple times to send orders more than once. This way guests can start with drinks and appetizer, move to entrée and then dessert in multiple sent orders. There will be an option with each menu item a guest orders to modify the item. This could be a text field or combination of text-field and list of common modifications (i.e., no mayo, sauce on side).

6.3.1.3.4 Hosting staff UI

This will have the same top-down view of the tables as the manager’s interface shows. With frequent automatic page refreshes, the hosting staff can stay up to date on the status of tables. The hosting staff will have the ability to tap/click on tables to modify the table’s status (i.e., Seated, Unseated, Needs Bussing). It will be the job of the hosting staff to check tables periodically to update their status, so they know where to take guests based on a server seating rotation. The hosting staff interface will also have a slidable list of servers in the form of a queue that shows which server is next in line to have a table seated. This can be overridden without a manager’s intervention in the case of a server in the queue already having a full section or the guests having a strong preference on where they want to sit. The queue of servers will be more of a reference.

* + - User Interface Wireframe

6.3.1.4

Graphical user interface

Description automatically generated

Figure 1- Manager UI (Draft wireframe image)

* 1. **Architecture** 
     + Event-driven architecture

6.4.1 CentRes will implement an event-driven architecture that utilizes time stamps for orders in and orders out. The application will be built based on four layers. Event Producer will sense the event (server opening the order creation page). Event channel will create a queue that handles events asynchronously and accepts the event (server placing order). Event processing engine will be the logic of the event and will decide an appropriate action to the event (in this case, the notification will need to be sent). Downstream event-driven activity will process the action of the event and trigger a notification of the order (notify line cook/chef/back of house).

* + - Data-centric architecture

6.4.2 CentRes will also implement a database-centric architecture that will utilize a database to track menu items, prices, and inventory.

* 1. **Database** 
     + CentRes will be using a MySQL relational database to manage orders, and employee information. MySQL can improve the efficiency of the database. Additionally, it can streamline the productivity of the restaurant, so that workers or employees will put in their effort to ensure that the business achieves their target goal and improves the revenue base.
     + Entity Relationship Diagram

Diagram

Description automatically generated

Figure 2- Entity Relationship Diagram (Draft)

* 1. **Top-level Classes** 
     + Specifications
       1. Main: In charge of login to access other classes.
       2. Manager: The highest level that can manage the waiter and cook. Manager can use all the provided functions that waiter and cook have as well.
       3. Waiter: Waiter has the second highest number of functions along with the ability to edit the table class. The waiter class can view the menu, edit orders, view orders, place orders, show order times it can also print an itemized bill of the orders.
       4. Cook: The cook has the lowest number of functions provided. The cook can view orders, edit orders, mark orders as ready, and view the menu.
       5. Tables: Tables class can display the tables in the restaurant, mark tables as seated, and display what tables need to be cleaned. The table class doesn’t change any of the preexisting values but oversees displaying the variables in an easy-to-read format.
       6. Class Diagram

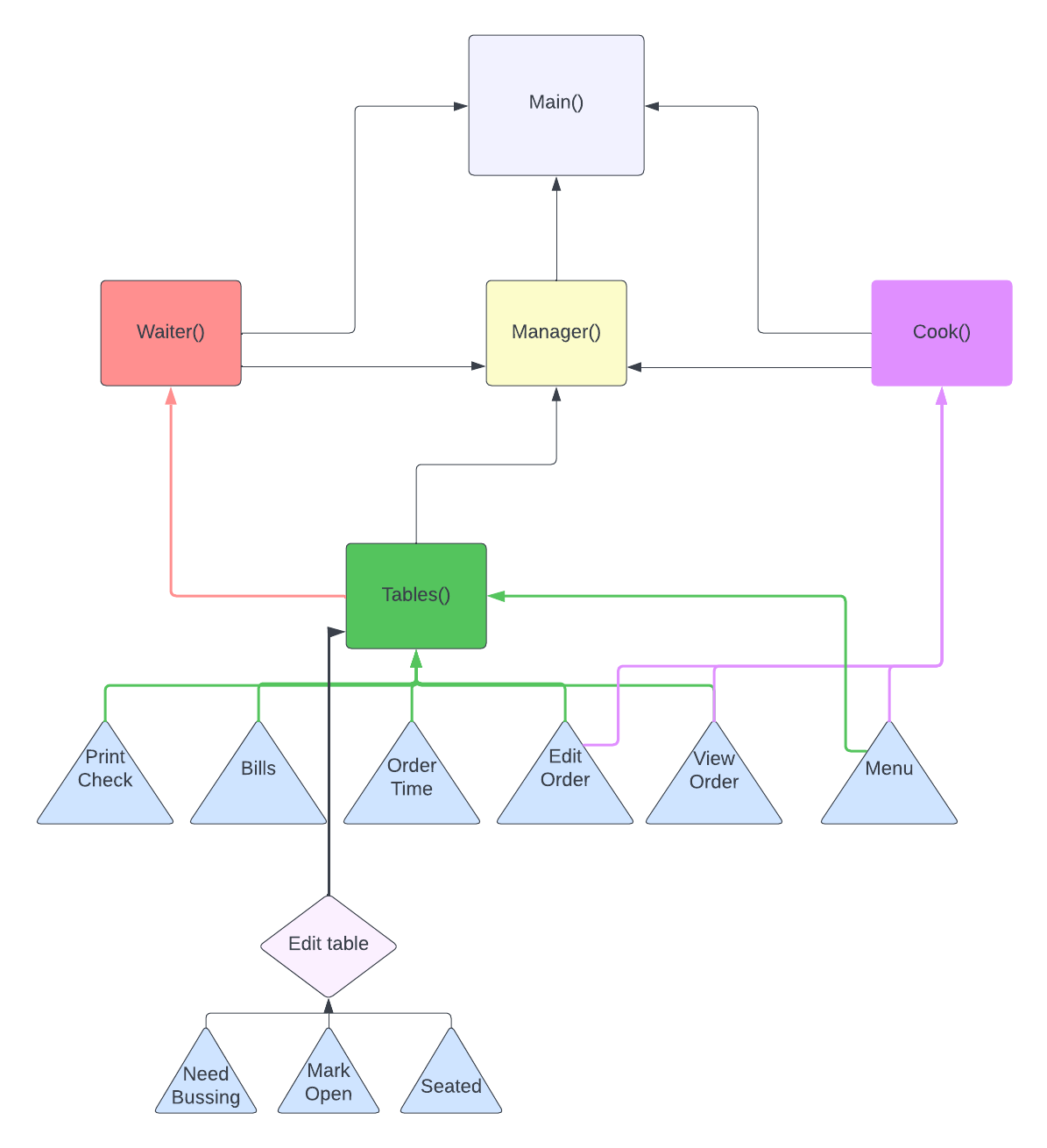


Figure 3- Class Diagram (Draft)

**6.7 Data Flow and States**

* + - Specifications

This is the second level Data Flow Diagram (DFD) which provide more detail of category such as employee and orders.

* + - The diagram has been used for the visualization of data processing and structured design of the restaurant process and working flow.
    - Data Flow Diagram

Diagram

Description automatically generated

Figure 4- Data Flow Diagram

* 1. **Reports** 
     + Manager
       1. Financial report generated for monthly and yearly tracking

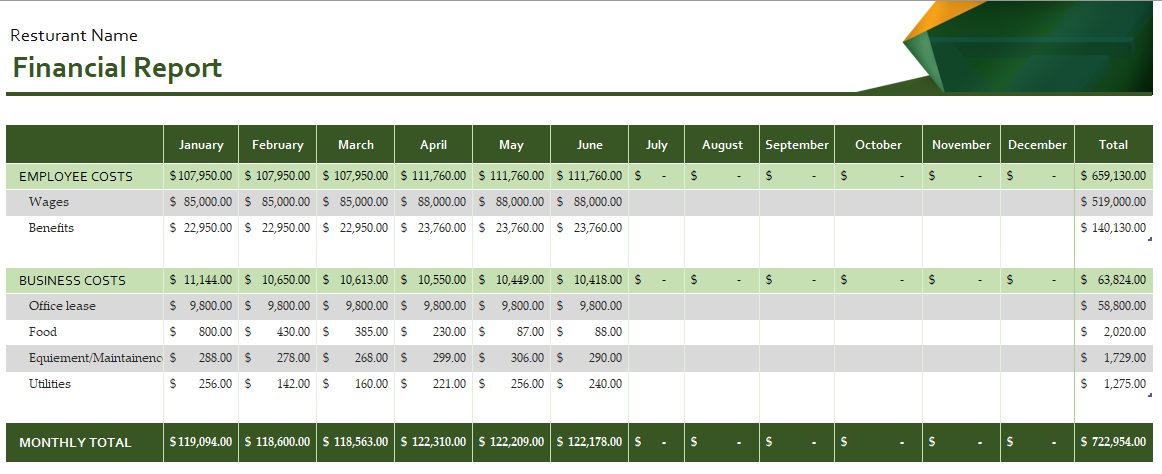


Figure 5- Financial Report Example

* + - 1. Inventory report on raw ingredients

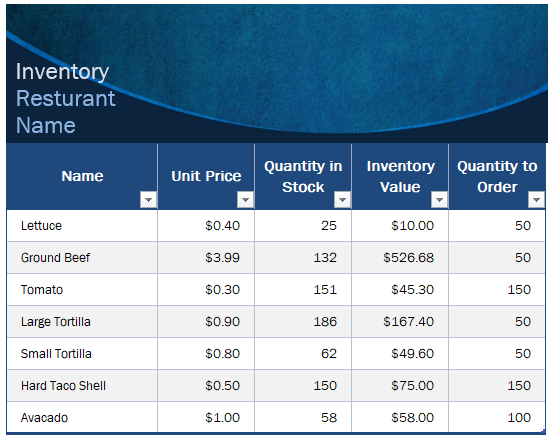


Figure 6-Inventory Report Example