***Software Engineering***

***Software Requirements Specification***

***(SRS) Document***

**Wan-Ten-Soup: Restaurant Manager**

**November 27, 2022**

**Final Version**

**By: Ewan Allen, Atsoupe Bessou Kpeglo, & Kristin Cattell**

**We Have Abided by the UNCG Academic Integrity Policy**

**Table of Contents**

[**1. Introduction**](#_heading=h.gjdgxs) **3**

[1.1. Purpose](#_heading=h.30j0zll) 3

[1.2. Document Conventions](#_heading=h.1fob9te) 3

[1.3. Definitions, Acronyms, and Abbreviations](#_heading=h.3znysh7) 3

[1.4. Intended Audience](#_heading=h.2et92p0) 3

[1.5. Project Scope](#_heading=h.tyjcwt) 4

[1.6. Technology Challenges](#_heading=h.3dy6vkm) 4

[1.7. References](#_heading=h.1t3h5sf) 4

[**2. General Description**](#_heading=h.4d34og8) **4**

[2.1. Product Perspective](#_heading=h.2s8eyo1) 4

[2.2. Product Features](#_heading=h.17dp8vu) 4

[2.3. User Class and Characteristics](#_heading=h.3rdcrjn) 5

[2.4. Operating Environment](#_heading=h.26in1rg) 5

[2.5. Constraints](#_heading=h.lnxbz9) 5

[2.6. Assumptions and Dependencies](#_heading=h.35nkun2) 5

[**3. Functional Requirements**](#_heading=h.1ksv4uv) **5**

[3.1. Primary](#_heading=h.44sinio) 5

[3.2. Secondary](#_heading=h.2jxsxqh) 5

[3.3. Use Cases](#_heading=h.z337ya) 6

[3.3.1. Actor: Manager (Kristin Cattell)](#_heading=h.3j2qqm3) 6

[3.3.2. Actor: Employee (Atsoupe Bessou)](#_heading=h.1y810tw) 6

[3.3.3. Actor: Customer (Ewan Allen)](#_heading=h.4i7ojhp) 7

[**4. Technical Requirements**](#_heading=h.1ci93xb) **8**

[4.1. Operating System and Compatibility](#_heading=h.3whwml4) 8

[4.2. Interface Requirements](#_heading=h.2bn6wsx) 8

[4.2.1. User Interfaces](#_heading=h.qsh70q) 8

[4.2.2. Hardware Interfaces](#_heading=h.3as4poj) 8

[4.2.3. Communications Interfaces](#_heading=h.1pxezwc) 8

[4.2.4. Software Interfaces](#_heading=h.49x2ik5) 8

[**5. Non-Functional Requirements**](#_heading=h.2p2csry) **8**

[5.1. Performance Requirements](#_heading=h.147n2zr) 8

[5.2. Safety Requirements](#_heading=h.3o7alnk) 9

[5.3. Security Requirements](#_heading=h.23ckvvd) 9

[5.4. Software Quality Attributes](#_heading=h.ihv636) 9

[5.4.1. Availability](#_heading=h.32hioqz) 9

[5.4.2. Correctness](#_heading=h.1hmsyys) 9

[5.4.3. Maintainability](#_heading=h.41mghml) 9

[5.4.4. Reusability](#_heading=h.2grqrue) 10

[5.4.5. Portability](#_heading=h.vx1227) 10

[5.5. Process Requirements](#_heading=h.3fwokq0) 10

[5.5.1. Development Process Used](#_heading=h.1v1yuxt) 10

[5.5.2. Time Constraints](#_heading=h.4f1mdlm) 10

[5.5.3. Cost and Delivery Date](#_heading=h.2u6wntf) 10

[5.6. Other Requirements](#_heading=h.19c6y18) 11

[5.7. Use-Case Model Diagram](#_heading=h.3tbugp1) 11

[5.8. Use-Case Model Descriptions](#_heading=h.28h4qwu) 11

[5.8.1. Actor: Manager (Kristin Cattell)](#_heading=h.nmf14n) 11

[5.8.2. Actor: Employee (Atsoupe Bessou)](#_heading=h.37m2jsg) 12

[5.8.3. Actor: Customer (Ewan Allen)](#_heading=h.1mrcu09) 12

[5.9. Use-Case Model Scenarios](#_heading=h.46r0co2) 12

[5.9.1. Actor: Manager (Kristin Cattell)](#_heading=h.2lwamvv) 12

[5.9.2. Actor: Employee (Atsoupe Bessou)](#_heading=h.111kx3o) 14

[5.9.3. Actor: Customer (Ewan Allen)](#_heading=h.3l18frh) 15

[**6. Design Documents**](#_heading=h.4k668n3) **16**

[6.1. Software Architecture](#_heading=h.2zbgiuw) 16

[6.1.1. High-Level Database Schema](#_heading=h.3ygebqi) 19

[6.2. Software Design](#_heading=h.2dlolyb) 21

[6.2.1. State Machine Diagram: Manager (Kristin Cattell)](#_heading=h.sqyw64) 21

[6.2.2. State Machine Diagram: Employee (Atsoupe Bessou)](#_heading=h.3cqmetx) 22

[6.2.3. State Machine Diagram: Customer (Ewan Allen)](#_heading=h.1rvwp1q) 23

[6.3. UML Class Diagram](#_heading=h.4bvk7pj) 24

[6.3.1. UML Class Diagram: Manager](#_heading=h.2r0uhxc) 24

[6.3.2. UML Class Diagram: Employee](#_heading=h.1664s55) 25

[6.3.3. UML Class Diagram: Customer](#_heading=h.3q5sasy) 26

[6.3.4. UML Class Diagram: Item](#_heading=h.25b2l0r) 26

[6.3.5. UML Class Diagram: User](#_heading=h.ocjekxxh9nc1) 27

[6.3.6. UML Class Diagram: OrderItems](#_heading=h.ei70p1laiv2l) 27

[6.3.7. UML Class Diagram: Menu](#_heading=h.e56iigl5obti) 28

[6.3.8. UML Class Diagram: OrderMenu](#_heading=h.u2rrxsflqzf2) 28

[6.3.9. UML Class Diagram: Schedule](#_heading=h.m72m67kx8k9m) 29

[6.3.10. UML Class Diagram: ScheduleDate](#_heading=h.wrfgj19ur0dk) 29

[6.3.11. UML Class Diagram: Table](#_heading=h.b2pjci5weg95) 30

[6.3.12. UML Class Diagram: HomeController & API](#_heading=h.vlh2xamvam6i) 30

[6.3.13. UML Class Diagram: Security Configs](#_heading=h.v0tssw8o4r35) 30

[**7. Scenario**](#_heading=h.5g2ic9o9s9ug) **31**

[7.1. Brief Written Scenario with Screenshots](#_heading=h.b9pr8b808hqf) 31

# Introduction

## Purpose

The project we are developing is a restaurant POS System designed with the customer and ease of access in mind. Its purpose is that it is designed to reduce consumer wait time by allowing the customer to place their own order for food. Not only that, but the system also allows the employees to manage customers and their orders in an efficient way by providing an easy-to-use interface for the employees to manage and modify based on customer activity. Lastly, we know how difficult and stressful managing a restaurant can be, so we’ve also developed this system in order to allow managers to efficiently manage and modify the specifics of their restaurant. The system overall decreases the stress and pressures on managers and employees, while increasing the pleasure of customers!

## Document Conventions

The purpose of this Software Requirements Document (SRD) is to describe and present all of the requirements for the Restaurant Manager. There are three types of requirements for this system: customer-oriented, employee-oriented, and manager-oriented requirements, each describing the system from the perspective of either the customer, employee, or manager. Within the document itself, it will be formatted with a numerated point list system, allowing for easy access of each point and “chapter” that will explain in detail each requirement for our system. This makes it easier for us and the reader to follow.

## Definitions, Acronyms, and Abbreviations

| Java | A programming language originally developed by James Gosling at Sun Microsystems. We will be using this language to build the Restaurant Manager. |
| --- | --- |
| DB | An abbreviation for Database. |
| .HTML | Hypertext Markup Language. This is the code that will be used to structure and design the web application and its content. |
| SpringBoot | An open-source Java-based framework used to create a micro Service. This will be used to create and run our application. |
| MVC | Model-View-Controller. This is the architectural pattern that will be used to implement our system. |
| Spring Web | Will be used to build our web application by using Spring MVC. This is one of the dependencies of our system. |
| Thymeleaf | A modern server-side Java template engine for our web environment. This is one of the dependencies of our system. |
| NetBeans | An integrated development environment (IDE) for Java. This is where our system will be created. |
| API | Application Programming Interface. This will be used to implement a function within the software where the current date and time is displayed on the homepage. |

## Intended Audience

The following are the potential people who will read this document:

* The creators of the document: Ewan Allen, Atsoupe Bessou Kpeglo, and Kristin Cattell
  + For these people, the entire document is intended to be read and edited if need be.
* Sunny Ntini
  + For this person, the entire document is intended to be read in order to make sure that all requirements are fulfilled for the project and to provide feedback on any of the functions and requirements of the system.
* The CSC 340-01 Fall 2022 class and anyone else who may or may not need to use this system
  + For these people, it is intended for them to only read the introduction and general description of the system. However, they may also read the functional and non-functional requirements if they would like to provide feedback on those requirements.

## Project Scope

The goal of the software is to provide an easy-to-use interface for all customers, employees, and managers of a restaurant, as well as provide customers with flexibility to meet their needs. This aligns with the overall business goals of a restaurant as a restaurant requires fast and efficient service in order to fulfill the needs of its customers.

The benefits of the project to business include:

* Relieving stress and pressure from employees and managers as customers are given the opportunity to request services when needed.
* Increasing pleasure to customers as they are given more power when they want to order rather than having to wait for an employee to ask for their order.
* Reducing the amount of time that a customer needs to wait; therefore, increasing the amount of customers that are able to be served in the restaurant within a day.

## Technology Challenges

The technology used with our project must have at least Java JDK 17 or above in order to execute and run the system, especially since SpringBoot generates the project to be Java 17.

## References

Ntini, S. (2022). *Use-Case Models* [PowerPoint presentation]. Canvas. https://uncg.instructure.com/

Ntini, S. (2022). *MVC Architecture* [PowerPoint presentation]. Canvas. https://uncg.instructure.com/

# General Description

## Product Perspective

After a week and a half of difficult decision making within our capabilities, we decided upon developing a system for a restaurant, as it could work great leaning into the criteria of the project as well as the utility it may provide.

## Product Features

Summary of the functions the system would perform and features to be included:

* Customization: Customers are allowed to manage and modify their orders in any way that they desire.
* Inventory Management: Managers can see the inventory that the restaurant currently has and can easily request to local vendors any products that need to be restocked.
* Table Management: Employees can manage and modify which tables within the restaurant are occupied.
* Service Requests: Customers can easily request for any services, whether it be asking an employee for help or requesting for the receipt. Customers can either use cash or card to process their payment.
* Employee Management: Managers can manage their employees by removing or adding any employees within the system, while also being able to modify an employee’s schedule as well.

## User Class and Characteristics

Our web application does not require our users to have any prior knowledge of Java or any knowledge on how to operate SpringBoot. The only knowledge that the users need to know, if they are a customer, is the basic operation of a web application. If the user is an employee or a manager, then they should have knowledge of the food products that are being served to the customers.

## Operating Environment

The environment that the software is being designed to operate in is the Java Runtime Environment, where JDK 17 is the minimum version required, and SpringBoot.

## Constraints

Due to the use of the programming language, Java, to create the software, the operating environment needs to have Java 17 (major version 61) or above installed in order for the software to properly execute and run.

## Assumptions and Dependencies

The software will be dependent on Spring Web and Thymeleaf in order to create and execute the MVC architecture that will be developed within NetBeans. The application will also use the World Time API (http://worldtimeapi.org/) that will display the current date and time on the home dashboard for everyone to see.

# Functional Requirements

## Primary

The following are the primary responsibilities of the system:

* The system will provide customers with a list of food products that are available for the customer to order.
* The system will allow customers to customize and place their orders.
* The system will allow customers to request services from the employees.
* The system will allow employees to see which tables are available and assign them to arriving customers.
* The system will allow employees to see the orders of each table.
* The system will allow managers to customize product offerings to customers.
* The system will allow managers to manage and modify an employee’s schedule.
* The system will allow managers to manage and modify produce stocks.
* The system will send product restock information directly to vendors.

## Secondary

The following are the secondary functions to support the primary responsibilities of the system:

* Password protection for information only accessible to employees, managers, and each individual table.
* Protection scheme so that customers can only alter and see their orders and not other customers’ orders.
* The application will utilize the World Time API to display the current date and time on the home dashboard for all users to see.
* Thymeleaf functions within HTML coding will help display all items in their corresponding tables accurately.

## Use Cases

### Actor: Manager (Kristin Cattell)

| Name of Use Case: | Manage Produce |
| --- | --- |
| Summary: | In order to manage food product stocks, the manager selects specific food products and places orders for restock. |
| Basic Flow: | 1. The use case starts when the user indicates that they want to manage and modify how much produce the restaurant has. 2. The system displays all current products that the restaurant currently has and how much of each product is in stock. 3. The user selects which products that they want to restock. 4. The system requests from the user the amount of each product that they want to be restocked. 5. The user selects an amount for each product that they want to be restocked. 6. The system sends this restock request to a local vendor. |
| Alternative Flows: | None |

| Name of Use Case: | Modify Schedules |
| --- | --- |
| Summary: | In order to create schedules for employees, the manager modifies time slots for specific employees. |
| Basic Flow: | 1. The use case starts when a user indicates that they want to manage and modify an employee’s schedule for the current displayed week. 2. The system displays a list of employees that the restaurant currently employs as well as the current week. 3. The user selects which employee that they want to manage and modify the schedule of. 4. The user enters the schedule of an employee for a specific week. 5. The system stores any new changes to an employee's schedule. |
| Alternative Flows: | Step 4: If there’s already a time occupying a selected date, then the system allows the user to modify that time slot. |

### Actor: Employee (Atsoupe Bessou)

| Name of Use Case: | Manage Table Availability |
| --- | --- |
| Summary: | In order to assign tables to specific customers, the employee will select a specific table for the customers to be seated at. |
| Basic Flow: | 1. The use case starts when a user indicates that they want to assign a table to a specific set of customers. 2. The system displays a list of tables and indicates which tables are empty or occupied. 3. The user selects a non-empty table for the customers to be seated at. 4. The system stores any new inputs to any of the tables within the list. |
| Alternative Flows: | Step 3: If all the tables are occupied, but an occupied table’s customers have already left, the system can allow the employee to modify that occupied table slot and change its status for the incoming customer(s). |

| Name of Use Case: | View Orders |
| --- | --- |
| Summary: | In order to access customers’ orders, the employee will select a specific table and the system will display their current orders. |
| Basic Flow: | 1. The use case starts when a user indicates that they want to access the orders created by a specific table. 2. The system displays a list of tables in the restaurant and indicates the tables’ statuses. 3. The user selects which table they want to see the orders of. 4. With a specific table selected, the system displays a list of orders that that table has ordered. 5. The user applies changes to orders as to whether they want to change the quantity of an ordered item or to delete specific orders off that list. 6. The system saves these changes to the orders. |
| Alternative Flows: | None |

### Actor: Customer (Ewan Allen)

| Name of Use Case: | Place Orders |
| --- | --- |
| Summary: | In order for a customer to place their order, they must select from a list of products available on the menu. |
| Basic Flow: | 1. The use case starts when a user indicates that they want to create an order. 2. The system displays a list of products available for the customer to order. When a specific item is selected, the system requests the quantity of how much the customer would like of that order. 3. The user selects which food products that they would like to order, as well as the quantity of such products. 4. The system saves the order and displays the order for the employees to view and manage. |
| Alternative Flows: | Step 3: If a user would like to delete a food product from their order, they can click the trash icon and the system will no longer display that product as one that the user has selected to order. |

### 

| Name of Use Case: | Call for Service |
| --- | --- |
| Summary: | In order for a customer to request for services from an employee, they must select the service they want help with. |
| Basic Flow: | 1. The use case starts when a user indicates that they want to request service from an employee. 2. The system lets the customer know that a service request has been made. 3. The system sends the request for the employees to view. |
| Alternative Flows: | None |

# Technical Requirements

## Operating System and Compatibility

* Windows, macOS, and Linux operating systems, basically any system that can run Java.
* The software is also compatible with any system that has JDK 17 or above installed.

## Interface Requirements

### User Interfaces

* User access will be defined into three categories and levels of access. The user will be either a manager, a customer, or an employee. However, any user may view the menu at any point while on the web application.

### Hardware Interfaces

* Any computer capable of running Java is all the project requires.
* The only current network requirement is that the software must have internet connection.
* A communication protocol that will be used within the software is HTTP in order for the software to call the World Time API to display the current date and time.

### Communications Interfaces

* It must be able to connect to the internet as well as the local database on phpMyAdmin.
* The communication protocol, HTTP, must be able to connect to the World Time API and return the current date and time.

### Software Interfaces

* SpringBoot will interact with the Java project and HTML/CSS/JavaScript files that create and design the application in order to execute and display our application within a web application.
* The manager-oriented interface must be able to interact with the DB in order to manage employee information and inventory.
* The employee-oriented and customer-oriented interface must be able to interact with the DB in order to view menu items and orders of each specific table. Note: Customers are only allowed to view orders and menu items from their specific table login only.

# Non-Functional Requirements

## Performance Requirements

The following are performance requirements for the application:

* The novice user (manager) will be able to manage and modify product inventory in less than 10 minutes.
* The expert user (manager) will be able to manage and modify product inventory in less than 5 minutes.
* The novice user (employee) will be able to manage and modify table occupation in less than 3 minutes.
* The expert user (employee) will be able to manage and modify table occupation in less than 1 minute.

## Safety Requirements

The following are safety requirements for the application:

* Customer verification will be required to discourage and attempt trolling of orders, or table hopping and overflowing the application with false orders.
* Employees will be able to access a table’s status, such as a table being ready to pay with cash or showing that a table has already paid with a card to make sure that each table has actually paid before leaving.

## Security Requirements

The following are security requirements for the application:

* The application will only be usable by authorized users, which include the customers, the employees, and the manager.
* In order to access the application from the manager’s perspective, the manager must enter their username and password to access the application that contains crucial, sensitive information about the products and employees.
* In order to access the application from the employee’s perspective, the employee must enter their username and password to access their own personal employee information and abilities.
* In order for customers to access the ability to make orders and view the items that they’ve already ordered, an employee will login each customer to a specific table login account based on the table that they’re currently seated at. For example, customers seated at table 1 will be logged into table 1’s account to access menu items and order items only for table 1.

## Software Quality Attributes

### Availability

* The application is available to use at any time that the restaurant is open for customers and employees. However, managers are allowed to operate the application any time during the day to manage inventory.

### Correctness

* Every order that the customer places will always be displayed correctly for the employees to view and manage.
* The number of products within the inventory will always be accurate to what’s currently available in the restaurant.
* Each table displayed in the list of tables for employees will always correctly show whether or not a table is occupied.

### Maintainability

* If a customer wants to delete a food product from their order before sending it, they can easily do so by clicking on the trash icon next to that item in their order.
* If the manager makes a mistake on scheduling employees, they can easily go and alter an employee’s schedule.
* If there are currently not enough ingredients to make a food product, the manager can easily remove the food product on the menu for customers until those ingredients have been restocked.

### Reusability

* The application can be easily designed so that it can be reused and implemented in many different restaurants.

### Portability

* The application can be operated on any operating system that supports Java, including Windows, macOS, and Linux.

## Process Requirements

### Development Process Used

* The agile incremental development process will be used to develop the application.

### Time Constraints

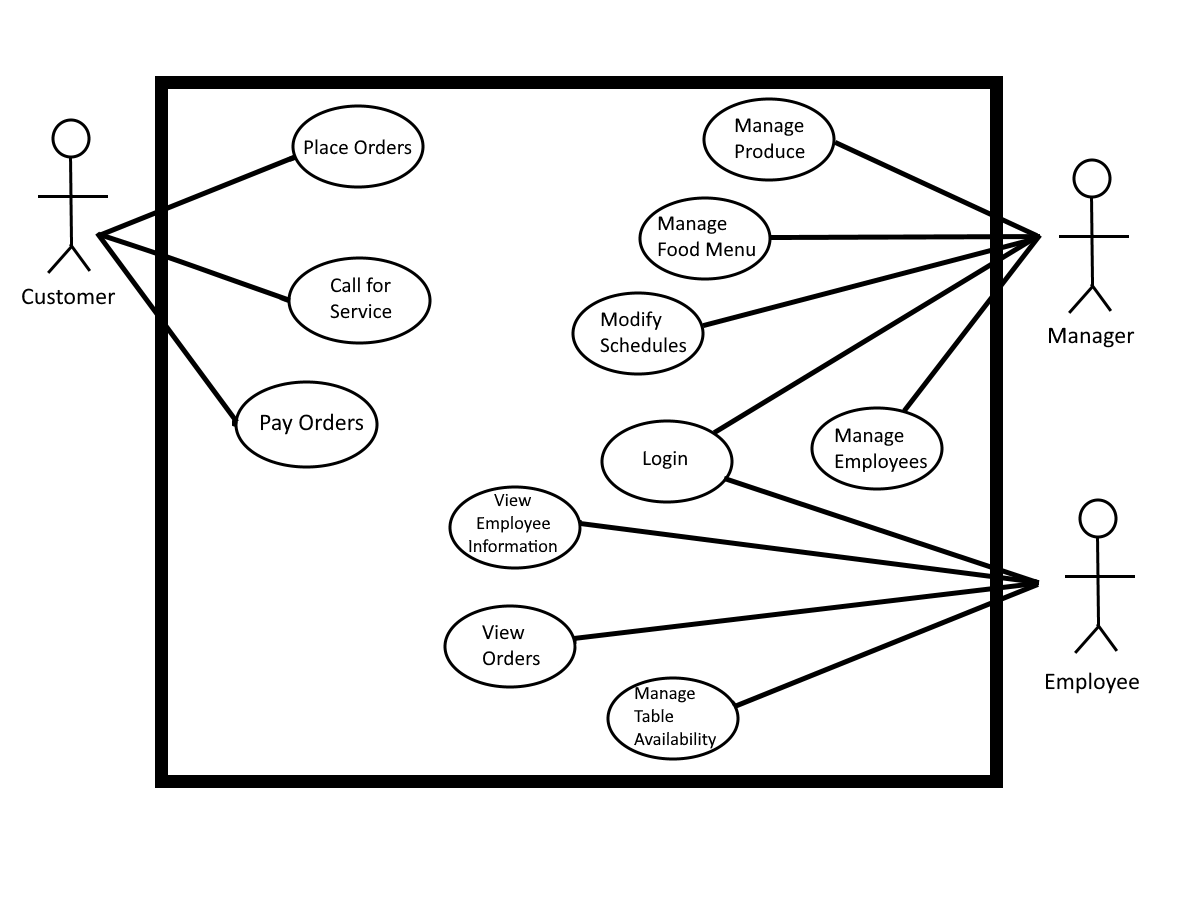
* The design document for the application is to be completed by October 4, 2022.
* The prototype of the application is to be completed by October 18, 2022.

### Cost and Delivery Date

* There will be no monetary cost to create and design the application.
* The final, completed version of the application will be delivered by November 15, 2022, which is the first day each group will start presenting their final applications.

## Other Requirements

## Use-Case Model Diagram



## Use-Case Model Descriptions

### Actor: Manager (Kristin Cattell)

* **Login**: The manager will be able to login into the system using the credentials provided by admins. After logging into the system, the manager can view different tabs that will lead them to view details either about the stock of produce, the food menu, and the employees within the establishment.
* **Manage Produce**: The manager can view and manage the amount of stock for each food ingredient that the restaurant currently has. If there needs to be any products/ingredients that need to be restocked, then the manager can also order any amount of each product that they desire and the system will send that request to a local vendor.
* **Manage Food Menu**: The manager can view each food and drink item that is on the menu. The manager can also modify which food/drink items are on the menu while also being able to modify the prices of each item as well. Any modifications done to the menu will also update the food menu seen by customers and employees.
* **Modify Schedules**: The manager can view each schedule of any employee that they select. After selecting an employee, the manager can edit their schedule by modifying the time slot for particular days of the week. The time slots would be updated and the changes would also be displayed to the employees as well immediately after each update.
* **Manage Employees**: The manager can view each employee within the establishment and add/delete any employees that have been hired or fired. The manager can also manage and modify an employee’s information and pay rate as well. Any updates will be displayed to the employees to indicate any changes the manager has made to their information or pay rate.

### Actor: Employee (Atsoupe Bessou)

* **Login**: The employee will be able to log in the system with their password and username. After they login, they can view the dashboard and check the menu.
* **View Orders**:The employee can view customer orders and be able to make changes to orders when a customer makes a mistake on an already placed order.
* **Manage Table Availability**: The employee can access and view the status of each table.
* **View Employee Information**: The employee can view their information that’s within the database, also including their schedule and pay rate.

### Actor: Customer (Ewan Allen)

* **Place Orders**: The customer is given nearly dynamic control on what type of food they may order. Upon viewing the menu, the customer adds their food to a “cart” where it can be processed and sent back to the kitchen.
* **Call for Service**: The customer has the ability to summon an employee whenever they need help with anything. A waiter can be summoned if there is an error in the system, to place the order “the old-fashioned” way, as well as just general menu help, food complaints, or to ask for options that may not be present on the menu quite yet. Sometimes, it is also nice to just have another person to be speaking with.
* **Pay Orders**: The customer is given the ability to pay for their orders. If the customer is paying with cash, the system will call for an employee to assist with the payment.

## Use-Case Model Scenarios

### Actor: Manager (Kristin Cattell)

* **Login**:
  + **Initial Assumption**: The manager has a registered account to login into the system. The manager’s account information is saved in the database.
  + **Normal**: The manager will enter their username and password to log into their account.
  + **What Can Go Wrong**: The username and/or password that the manager enters does not match the username or password stored in the database.
  + **Other Activities**: The manager can ask a database admin to reset their password.
  + **System State on Completion**: The manager is logged in and can view the dashboard which has all the tabs that contain details on the employees, the food products, and the menu.
* **Manage Produce**:
  + **Initial Assumption**: The manager is logged into their account and can view their dashboard and navigate to the Manage Produce tab.
  + **Normal**: The manager will increase/decrease the amount of stock of any product/ingredient that the establishment has at a given time. If necessary, the manager will also send a request to a local vendor to restock on any chosen products.
  + **What Can Go Wrong**: The manager accidentally enters the wrong quantity for an item or places the wrong product in their order.
  + **Other Activities**: The manager can change the quantity of any item in their order and delete any items that they accidentally order from their cart.
  + **System State on Completion**: The inventory of the establishment is correctly updated based on what the manager counts at any given time. The system also has successfully sent a request to restock any products if the manager creates a request. The manager can also see this request on their dashboard in the Manage Produce tab.
* **Manage Food Menu**:
  + **Initial Assumption**: The manager has logged into their account and can view their dashboard and navigate to the Manage Food Menu tab.
  + **Normal**: The manager will update the food menu by adding/deleting any food/drink items or changing the price of any food/drink items.
  + **What Can Go Wrong**: The manager accidentally duplicates a food/drink item on the menu.
  + **Other Activities**: The manager can simply delete the duplicate food/drink item from the menu.
  + **System State on Completion**: The food menu is modified successfully to the manager’s content and any updates are now displayed on the menu within the Food Menu tab. The customers and employees can also see any updates by clicking on the Menu button that’s on almost every page of the web application.
* **Modify Schedules**:
  + **Initial Assumption**: The manager has logged into their account and can view their dashboard and navigate to the Modify Schedules tab.
  + **Normal**: The manager can select a certain employee that they want to change the schedule for. The manager can then modify time slots on any given day for the specified week.
  + **What Can Go Wrong**: The manager gives an employee the wrong times on a given time slot.
  + **Other Activities**: A time slot can be edited by clicking on the Update button so that the manager can edit and correct the times of an employee on any given day.
  + **System State on Completion**: The schedules of each employee that the manager selects is updated successfully to the manager’s content. The employees can also see any updates to their schedules on their dashboard in the View Employee Information tab.
* **Manage Employees**:
  + **Initial Assumption**: The manager has logged into their account and can view their dashboard and navigate to the Manage Employees tab.
  + **Normal**: The manager can add or delete any employees within the database. The manager can also select a certain employee that they want to modify, being able to edit an employee’s information or pay rate if need be.
  + **What Can Go Wrong**: The manager can accidentally give an employee the wrong information.
  + **Other Activities**: The manager can click on the Update button for any employee and update their information as many times as they want.
  + **System State on Completion**: The information of each employee that the manager modifies is updated successfully to the manager’s content. The employees can also see any updates to their information on their dashboard in the View Employee Information tab.

### Actor: Employee (Atsoupe Bessou)

* **Login**:
  + **Initial Assumption**: The employee should be able to login into the system with their username and passcode, view menu that are available, customer orders, and check what tables are taken or still ready to be used.
  + **Normal**: The employee will be able to enter their username and password to log in the system.
  + **What Can Go Wrong**: The employee can forget their password or username and not be able to log in the system.
  + **Other Activities**: The employee should ask the manager to help reset their password.
  + **System State on Completion**: The employee can log in the system and view the dashboard and the tabs that are available to them.
* **View Orders**:
  + **Initial Assumption**: The employee has logged into their account and can view their dashboard and navigate to the “Manage Tables and View Orders” tab.
  + **Normal**: The employee can view the orders placed by each table. The employee can also indicate whether an order has been completed or is still being made.
  + **What Can Go Wrong**: The employee can assign an order to be completed when it hasn’t actually been completed.
  + **Other Activities**: An order’s status can be reassigned by clicking on the Update button and updating the status of any table when need be.
  + **System State on Completion**: The order is viewed and its status is up-to-date and is successfully changed to Completed status when the order is fulfilled and given to the customer.
* **Manage Table Availability**:
  + **Initial Assumption**: The employee has logged into their account and can view their dashboard and navigate to the “Manage Tables and View Orders” tab.
  + **Normal**: The employee can assign and update the status of each table within the establishment to indicate whether it’s being occupied or not by customers.
  + **What Can Go Wrong**: A table is no longer occupied; however, it still hasn’t been cleaned or prepared yet for the next customers.
  + **Other Activities**: A table can be reassigned as Being Cleaned by modifying the status box for a table to indicate that the table is not fully prepared to be occupied by new customers.
  + **System State on Completion**: A table’s status is assigned successfully. It is updated and shows the accurate occupation at any given time. Any other employee can also see a table’s status on their dashboard in the “Manage Tables and View Orders” tab.
* **View Employee Information**:
  + **Initial Assumption**: The employee has logged into their account and can view their dashboard and navigate to the View Employee Information tab.
  + **Normal**: The employee can view their information, including their schedule and their pay rate.
  + **What Can Go Wrong**: Some of the information presented to the employee is inaccurate.
  + **Other Activities**: The employee can request for the manager to change/update their information in person.
  + **System State on Completion**: An employee has successfully viewed their information, whether it be general information, their schedule, or their pay rate..

### Actor: Customer (Ewan Allen)

* **Place Orders**:
  + **Initial Assumption**: The customer has been assigned to a table and can view their dashboard and navigate to the Place Orders tab.
  + **Normal**: The customer will create a list of food and drink items, indicating the amount of each item that they want to order. When finished ordering, the customer will submit their order request.
  + **What Can Go Wrong**: The customer wants to add more to their order.
  + **Other Activities**: The customer can create a new order by clicking on the Place Order button again. If the order hasn’t already been submitted, then the customer can delete a selection from their order by using the trash icon that’s displayed next to the selection.
  + **System State on Completion**: The order is created successfully. It is updated and displayed for the employees to see within the “Manage Tables and View Orders” tab of their dashboards.
* **Call for Service**:
  + **Initial Assumption**: The customer has been assigned to a table and can view their dashboard and navigate to the Call for Service tab.
  + **Normal**: The customer clicks on the Call for Service tab and the screen then indicates that a call for service has been placed.
  + **What Can Go Wrong**: The customer no longer needs help on something.
  + **Other Activities**: The customer can tell the employee that they no longer need help, thus allowing the employee to quickly change the table’s call of service status to “Okay” again.
  + **System State on Completion**: The customer has successfully placed a call for service. The system sends this call to the employees which can be viewed by the employees within the “Manage Tables and View Orders” tab.
* **Pay Orders**
  + **Initial Assumption**: The customer has been assigned to a table, has successfully submitted and received their orders, and can view their dashboard and navigate to the Pay Orders tab.
  + **Normal**: The customer enters in their credit/debit card information to pay for their meals. If they are paying with cash, the customer selects the Pay with Cash button and will call an employee to assist the customer.
  + **What Can Go Wrong**: The customers want to split the bill.
  + **Other Activities**: The customer can designate which orders they want to split at the cash register.
  + **System State on Completion**: The order is successfully paid for completely. The system updates and shows the employees the status of the table being paid for in the “Manage Tables and View Orders” tab.

# Design Documents

## Software Architecture

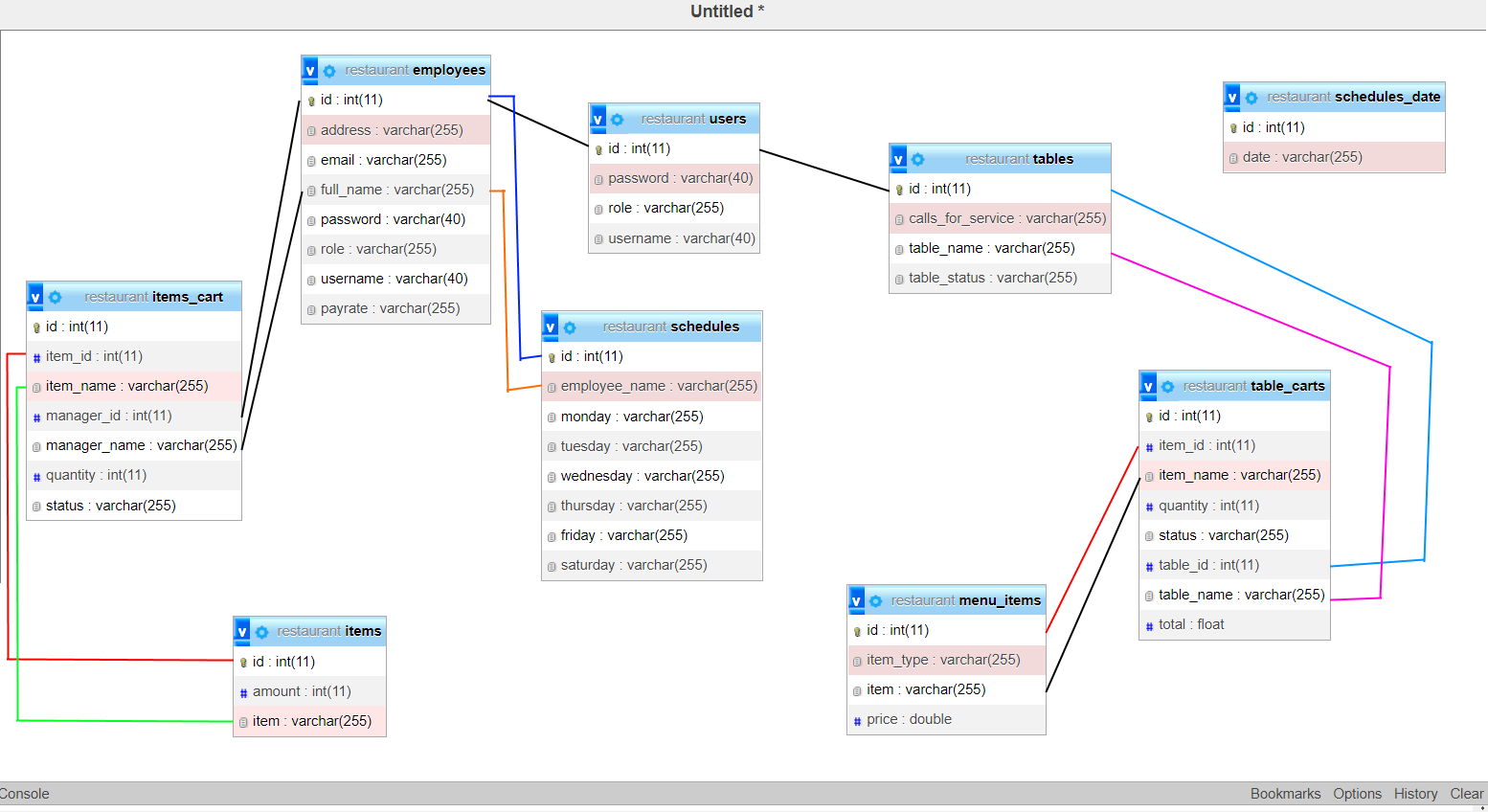
## 

1. **Security Config**
   * **Assumptions:** Assume that the Security Config classes look at the “users” table in the database and the attribute “role” to decide authorization to a person during login.
   * **AddOrderConfig:** The class that configures the system to only allow those with the proper authorization (their role being “MANAGER” in the “users” table in the DB) to access “/addNewOrder/” URL pages. Used to identify the current manager trying to order produce from a vendor.
   * **AddOrderTableConfig:** The class that configures the system to only allow those with the proper authorization (their role being "TABLE" in the "users" table in the DB) to access "/addNewOrderFromMenu/" URL pages. Used to identify the current table/customers trying to order from the menu.
   * **CustomerSecurityConfig:** The class that configures the system to only allow those with the proper authorization (their role being “TABLE” in the “users” table in the DB) to access “/customer/” pages. Used to identify tables and to separate each table so no table can view other tables’ orders.
   * **EmployeeSecurityConfig:** The class that configures the system to only allow those with the proper authorization (their role being “EMPLOYEE” in the “users” table in the DB) to access web pages that contain “/employee/” in the URL.
   * **ManagerSecurityConfig:** The class that configures the system to only allow those with the proper authorization (their role being “MANAGER” in the “users” table in the DB) to access web pages that contain “/manager/” in the URL.
2. **API**
   * **GetDateAPI:** The class that uses Gson to call the World Time API and parses out the current date and time.
   * **FormatDate:** The class that takes the parsed out date and time from the GetDateAPI class and formats that date to be readable from the home dashboard.
3. **Controllers**
   * **ItemController:** The class that contains all the HTTP request methods pertaining to displaying and modifying the table “items” in the DB, which are the items that the restaurant currently has in the inventory.
   * **ManagerController:** The class that contains all the HTTP request methods pertaining to displaying data from the DB accessible to managers and allowing the manager to do a variety of actions (modifying schedules, managing employees, etc).
   * **EmployeeController:** The class that contains all the HTTP request methods pertaining to displaying data from the DB accessible to employees and allowing the employee to do a variety of actions (managing the tables, managing order status, etc).
   * **CustomerController:** The class that contains all the HTTP request methods pertaining to displaying data from the DB accessible to customers and allowing the customer to do a variety of actions (ordering, paying, etc).
   * **HomeController:** The class that contains all the HTTP request methods pertaining to the home page that everyone sees when they first start using the web application. Also contains the HTTP request methods pertaining to the login screens. The class also uses methods in the GetDateAPI class to display the current date and time on the home dashboard.
   * **OrderItemsController:** The class that contains all the HTTP request methods pertaining to the manager ordering/restocking on produce ingredients/food or drink items sold at the restaurant. Also displays data from the DB accessible to managers that shows all the orders for produce done in the past.
4. **Models**
   * **Item Model:** The “Item” class creates the table “items” with the attributes that each entity will have.
     + **ItemService:** Interface class that contains abstract methods that interact with and modify the data stored in the “items” table in the DB
     + **ItemServiceImpl:** The class that implements the ItemService class and overrides all the abstract methods to create functional methods to interact with and modify data stored in the “items” table in the DB.
     + **ItemRepository:** The class that’s used by the item service layer to access the DB.
   * **User Model:** The “User” class creates the table “users” with the attributes that each entity will have. The table will contain the “roles” attribute for the Security Config classes to look at during login.
     + **CustomUserDetails:** The class that implements UserDetails (given by SpringBoot) and overrides methods dealing with user’s username/password and the login process.
     + **CustomUserDetailsService:** The class that implements UserDetailsService (given by SpringBoot) and overrides a method dealing with finding a user’s username in the database.
     + **Role:** An enum class that’s used to create an enum attribute classed “role” in the “users” table. The role is then used to be Security Config classes to decide proper login credentials and authorization.
     + **UserRepository:** The class that’s used by the user service layer to access the DB.
   * **Employee Model:** The “Employee” class creates the table “Employees” with the attributes that each entity will have.
     + **EmployeeService:** Interface class that contains abstract methods that interact with and modify the data stored in the “Employees” table in the DB. Will also have abstract methods to manage table availability, order status, etc.
     + **EmployeeServiceImpl:** The class that implements the EmployeeService class and overrides all the abstract methods to create functional methods to interact and modify the data stored in the “Employees” table. Will also override methods that will manage table availability, order status, etc.
     + **EmployeeRepository:** The class that’s used by the employee service layer to access the DB.
   * **Menu Model:** The “Menu” class creates the table “menu\_items” with the attributes that each entity will have.
     + **MenuService:** Interface class that contains abstract methods that interact with and modify the data stored in the “menu\_items” table in the DB.
     + **MenuServiceImpl:** The class that implements the MenuService class and overrides all the abstract methods to create functional methods to interact and modify the data stored in the “menu\_items” table.
     + **MenuRepository:** The class that’s used by the menu service layer to access the DB.
   * **OrderMenu Model**: The “OrderMenu” class creates the table “table\_carts” with the attributes that each entity will have. Used to implement the functionality of customers ordering from the menu.
     + **OrderMenuService:** Interface class that contains abstract methods that interact with and modify the data stored in the “table\_carts” table in the DB.
     + **OrderMenuServiceImpl:** The class that implements the OrderMenuService class and overrides all the abstract methods to create functional methods to interact and modify the data stored in the “table\_carts” table.
     + **OrderMenuRepository:** The class that’s used by the ordermenu service layer to access the DB.
   * **Schedule Model:** The “Schedule” class creates the table “schedules” with the attributes that each entity will have. Used to implement the functionality of managers creating and modifying the schedules of employees.
     + **ScheduleService:** Interface class that contains abstract methods that interact with and modify the data stored in the “schedules” table in the DB.
     + **ScheduleServiceImpl:** The class that implements the ScheduleService class and overrides all the abstract methods to create functional methods to interact and modify the data stored in the “schedules” table.
     + **ScheduleRepository:** The class that’s used by the schedule service layer to access the DB.
   * **ScheduleDate Model:** The “ScheduleDate” class creates the “schedules\_date” table with a single attribute and row to display the current week. Used to determine what week the schedules of each employee are for.
     + **ScheduleDateService:** Interface class that contains abstract methods that interact with and modify the data stored in the “schedules\_date” table in the DB.
     + **ScheduleDateServiceImpl:** The class that implements the ScheduleDateService class and overrides all the abstract methods to create functional methods to interact and modify the data stored in the “schedules\_date” table.
     + **ScheduleDateRepository:** The class that’s used by the scheduledate service layer to access the DB.
   * **OrderItems Model:** The “OrderItems” class creates the “items\_cart” table with the attributes that each entity will have. Used to implement the functionality of allowing managers to place orders for produce/ingredients used by the restaurant.
     + **OrderItemsService:** Interface class that contains abstract methods that interact with and modify the data stored in the “items\_cart” table in the DB.
     + **OrderItemsServiceImpl:** The class that implements the OrderItemsService class and overrides all the abstract methods to create functional methods to interact and modify the data stored in the “items\_cart” table.
     + **OrderItemsRepository:** The class that’s used by the orderitems service layer to access the DB.
   * **Table Model:** The “RestTables” class creates the “tables” table with the attributes that each entity will have. Used to implement the functionality of showing the status for each table for the employees to view and manage.
     + **TableService:** Interface class that contains abstract methods that interact with and modify the data stored in the “tables” table in the DB.
     + **TableServiceImpl:** The class that implements the TableService class and overrides all the abstract methods to create functional methods to interact and modify the data stored in the “tables” table.
     + **TableRepository:** The class that’s used by the table service layer to access the DB.
   * **Assumptions:**
     + Assume that the manager controller uses the Schedule, ScheduleDate, OrderItems, Item, Employee, and Menu Models to add/update/configure the schedules of employees, the date for the schedules, the orders for produce ingredients, the information of employees, and the menu.
     + Assume that the employee controller uses the Table Model to update and modify tables and their statuses/orders.
     + Assume that the customer controller uses the OrderMenu Model to allow customers to order from the menu and put their orders into the DB.
     + Assume the home controller uses the API classes to display the current date and time on the home dashboard.

## High-Level Database Schema

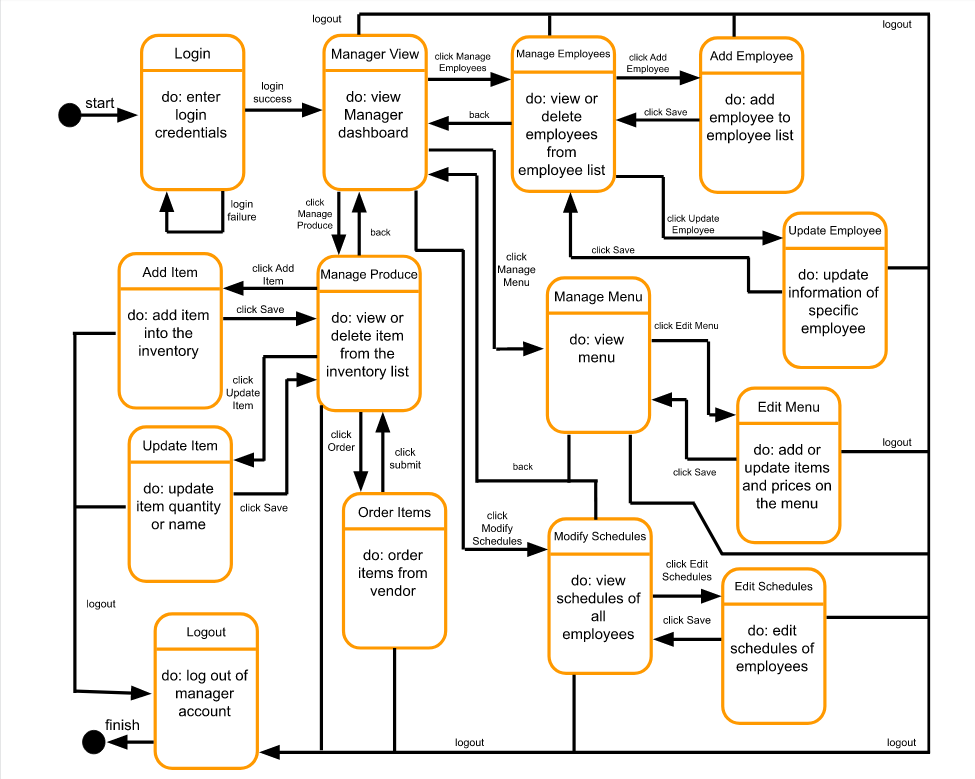
* **“items” Table:** Will store the item’s ID, item’s name, and the quantity of that item that’s currently in stock in the restaurant’s inventory.
* **“items\_cart” Table:** Will store the ID of each item order, the ID of the item that’s being ordered, the item’s name, the ID of the manager who’s ordering the item, the manager’s name, the quantity of the ordered item, and the status of whether or not the order of the item has been submitted yet or not.
* **“users” Table:** Will store every user's (managers, employees, and tables) ID, username, password, and role.
* **“employees” Table:** Will store every employee’s ID, full name, street address, email, username, password, pay rate, and role.
* **“schedules” Table:** Will store every employee’s ID and name associated with each schedule, as well as their schedules from Monday to Saturday.
* **“tables” Table:** Will store the id of each table and the tables name, as well as each table’s status (Occupied, Ordering, Paying, etc) and call for service status.
* **“menu\_items” Table:** Will store the id of each menu item, the menu item’s name, the type of menu item it is, and the price of each menu item.
* “**table\_carts” Table:** Will store the ID of each menu order from each customer, the ID of the menu item that’s being ordered, the menu item’s name, the quantity of ordered menu item, the status of that order, the ID of the table that’s ordering the menu item, the table’s name, and the total price from ordering that menu item.
* **“Schedules\_date” Table:** Table that contains only one row with the id and the date name of the date to be displayed for the employees to know which week their schedules are for.

**High-Level Database Schema:**

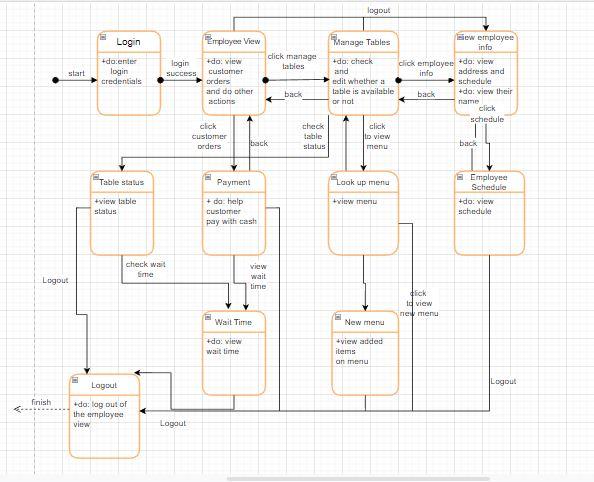


## Software Design

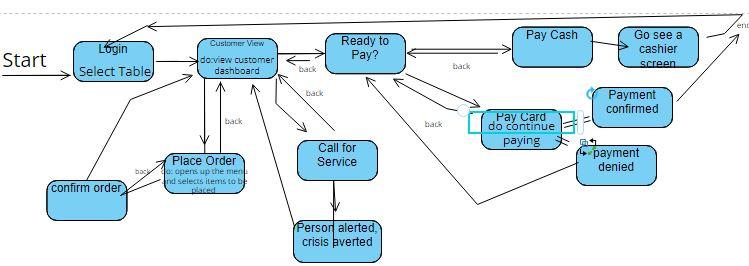
### State Machine Diagram: Manager (Kristin Cattell)



### State Machine Diagram: Employee (Atsoupe Bessou)

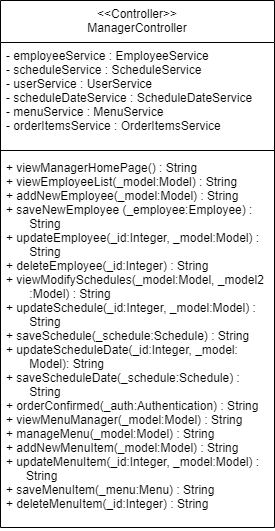


### State Machine Diagram: Customer (Ewan Allen)

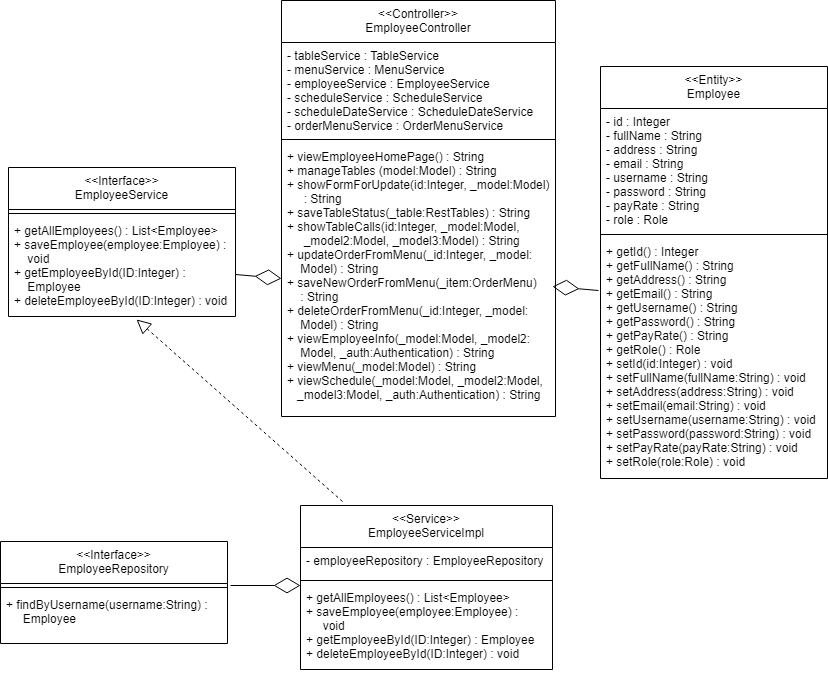


## UML Class Diagram

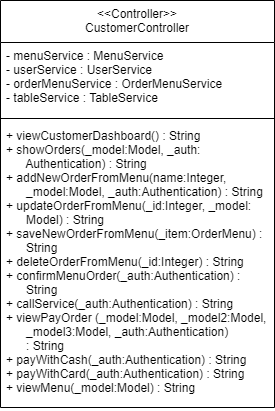
### UML Class Diagram: Manager



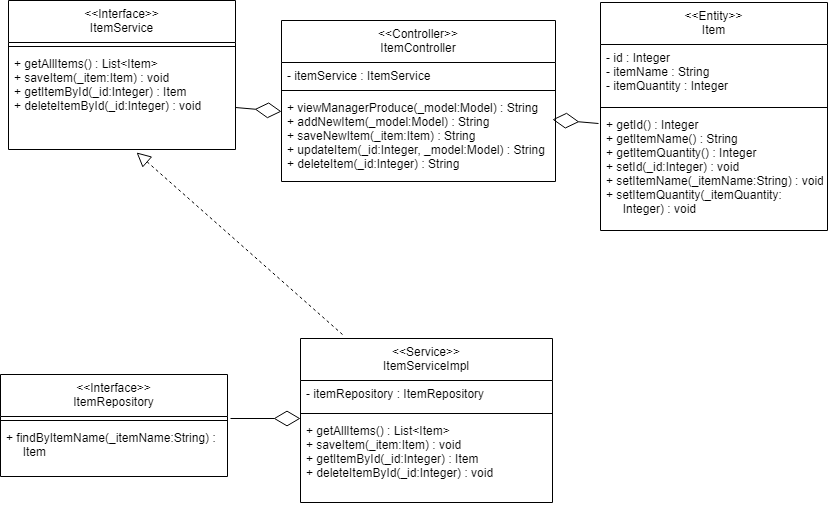
### UML Class Diagram: Employee



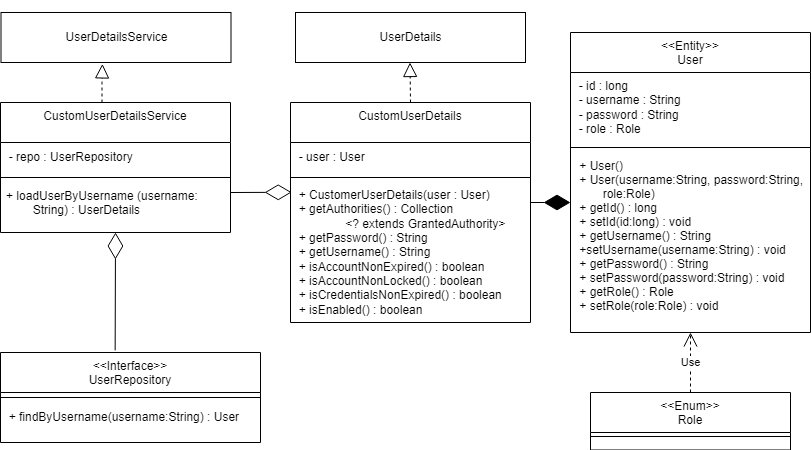
### UML Class Diagram: Customer



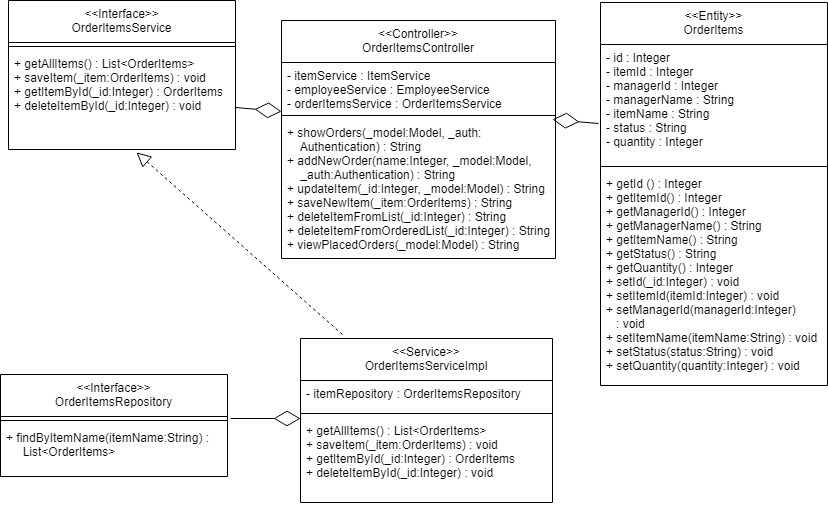
### UML Class Diagram: Item



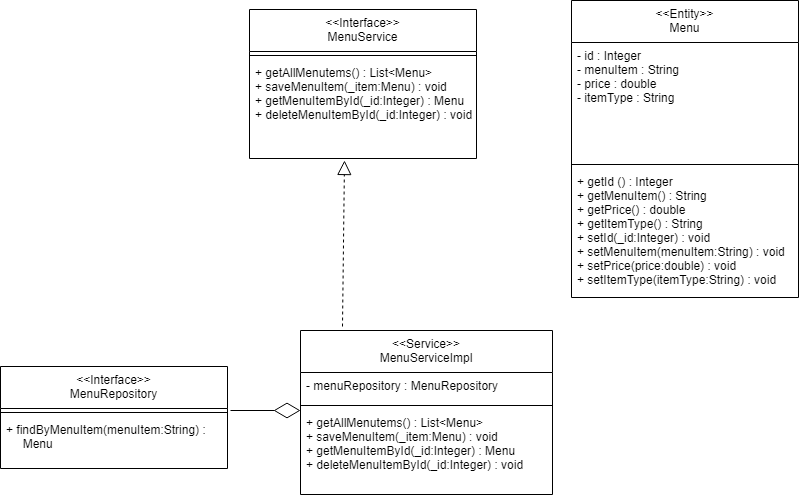
### UML Class Diagram: User



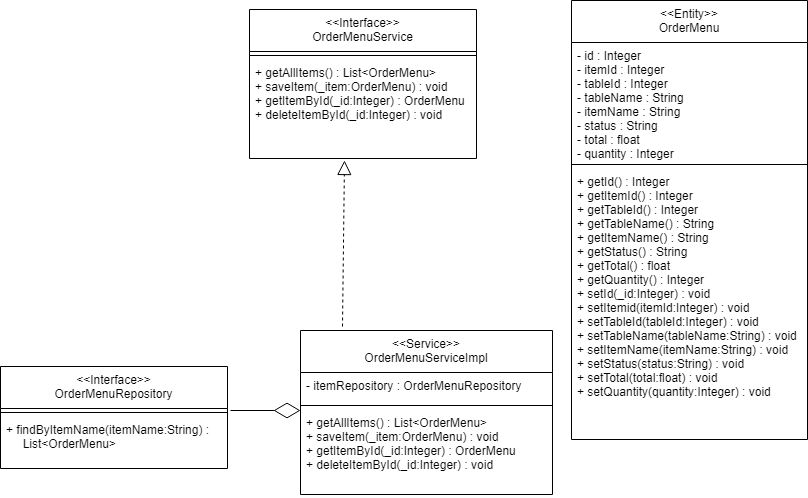
### UML Class Diagram: OrderItems

****

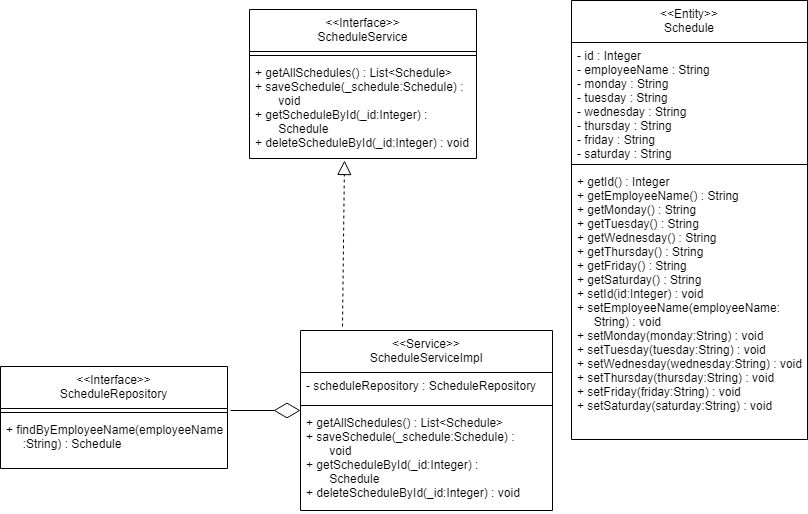
### UML Class Diagram: Menu

****

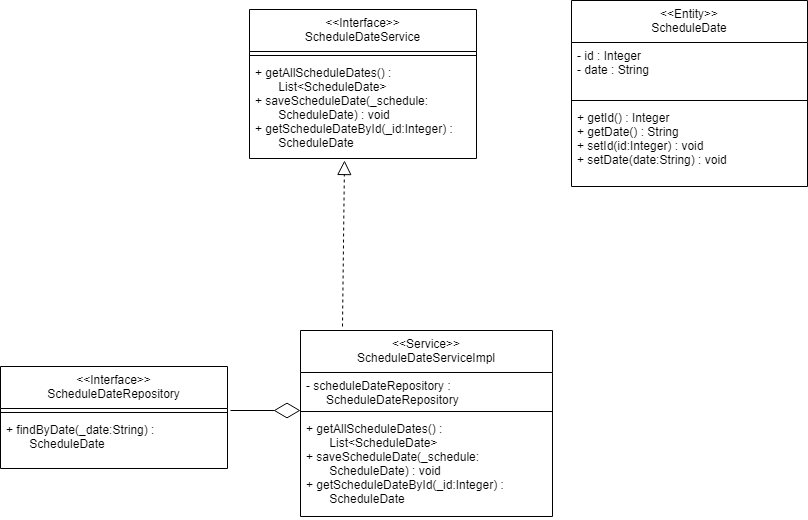
### UML Class Diagram: OrderMenu



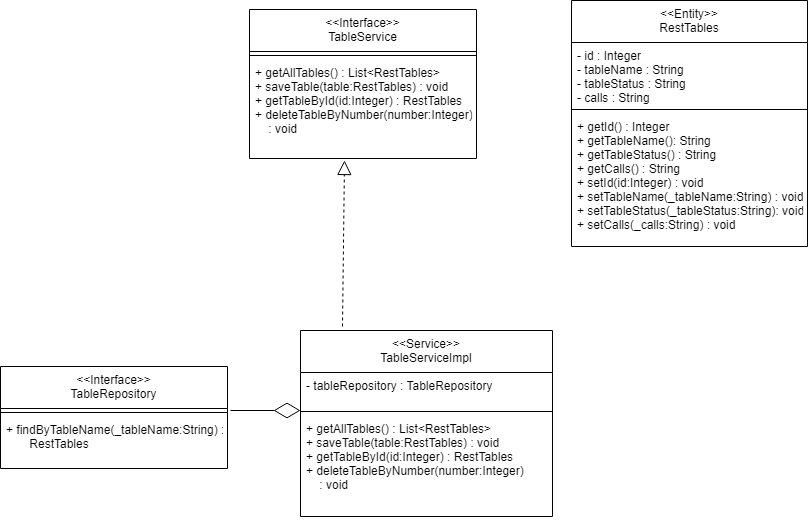
### UML Class Diagram: Schedule



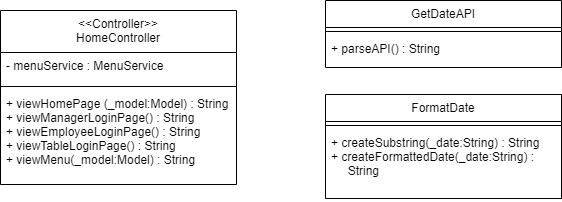
### UML Class Diagram: ScheduleDate



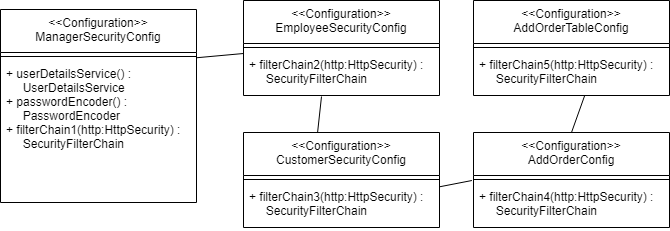
### UML Class Diagram: Table



### UML Class Diagram: HomeController & API



### UML Class Diagram: Security Configs

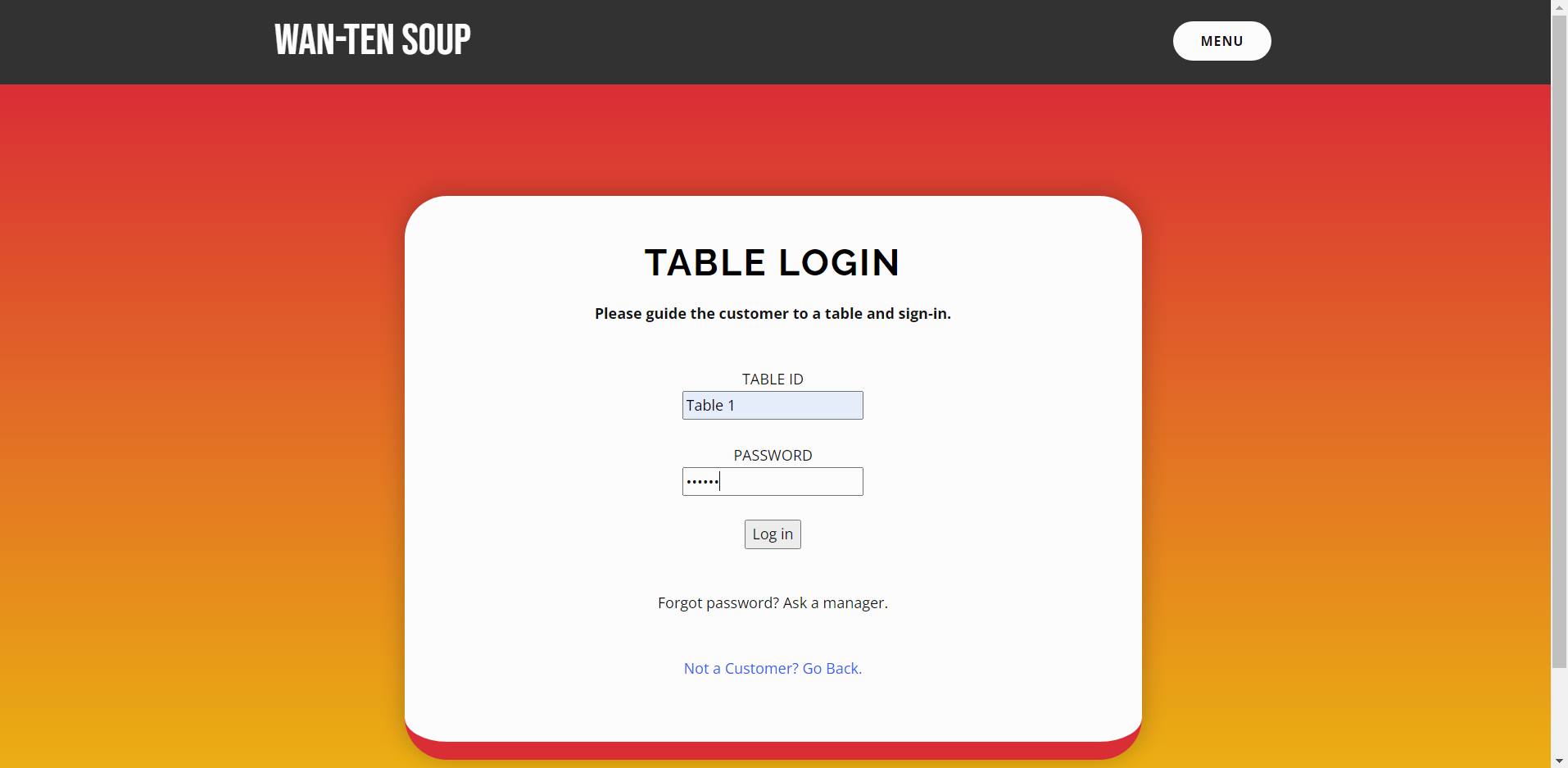


# Scenario

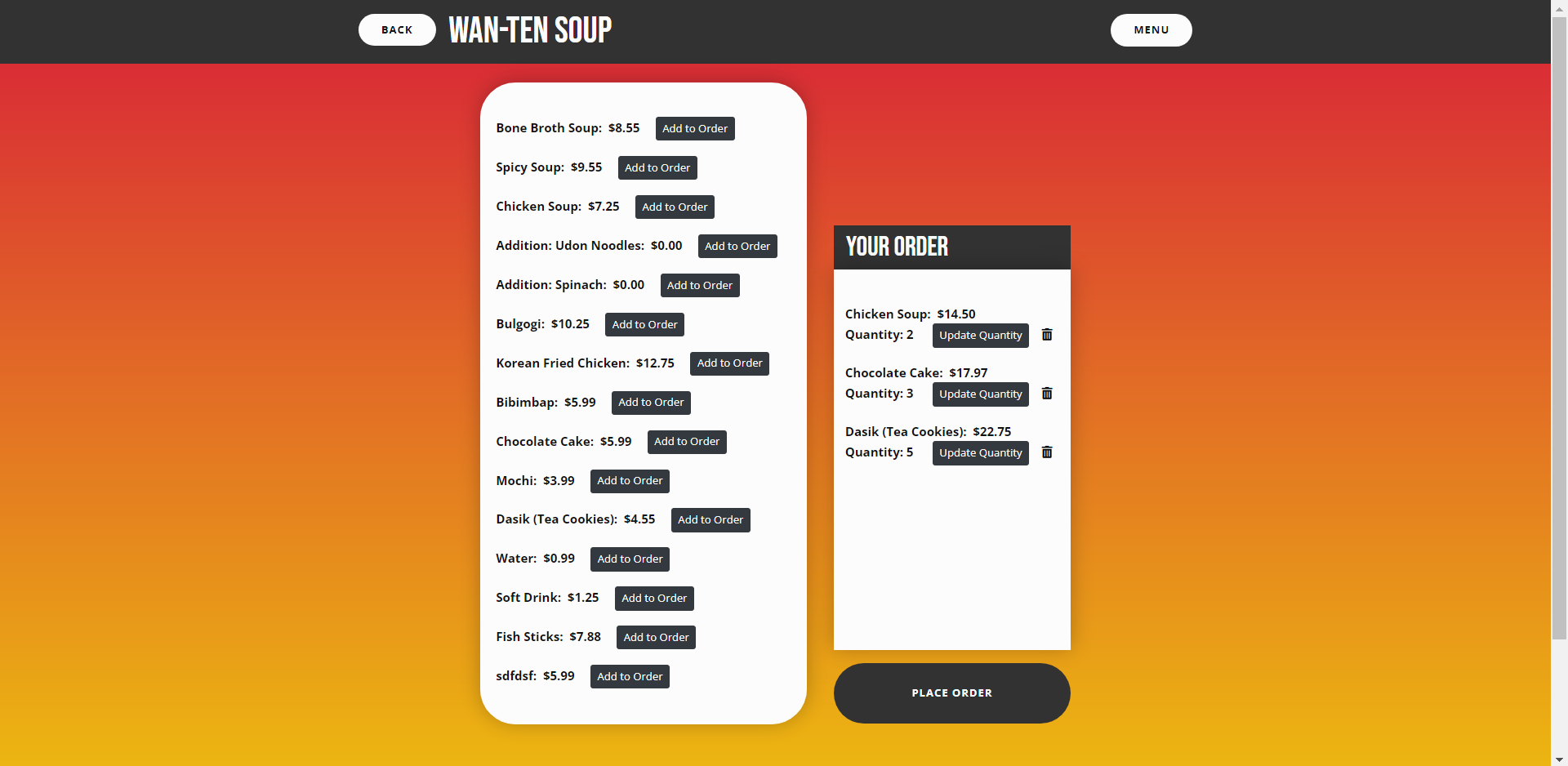
## Brief Written Scenario with Screenshots

*Customer: Place Order use case:*

1. Customer C1 is guided to a table and is logged in with the table's credentials by an employee.

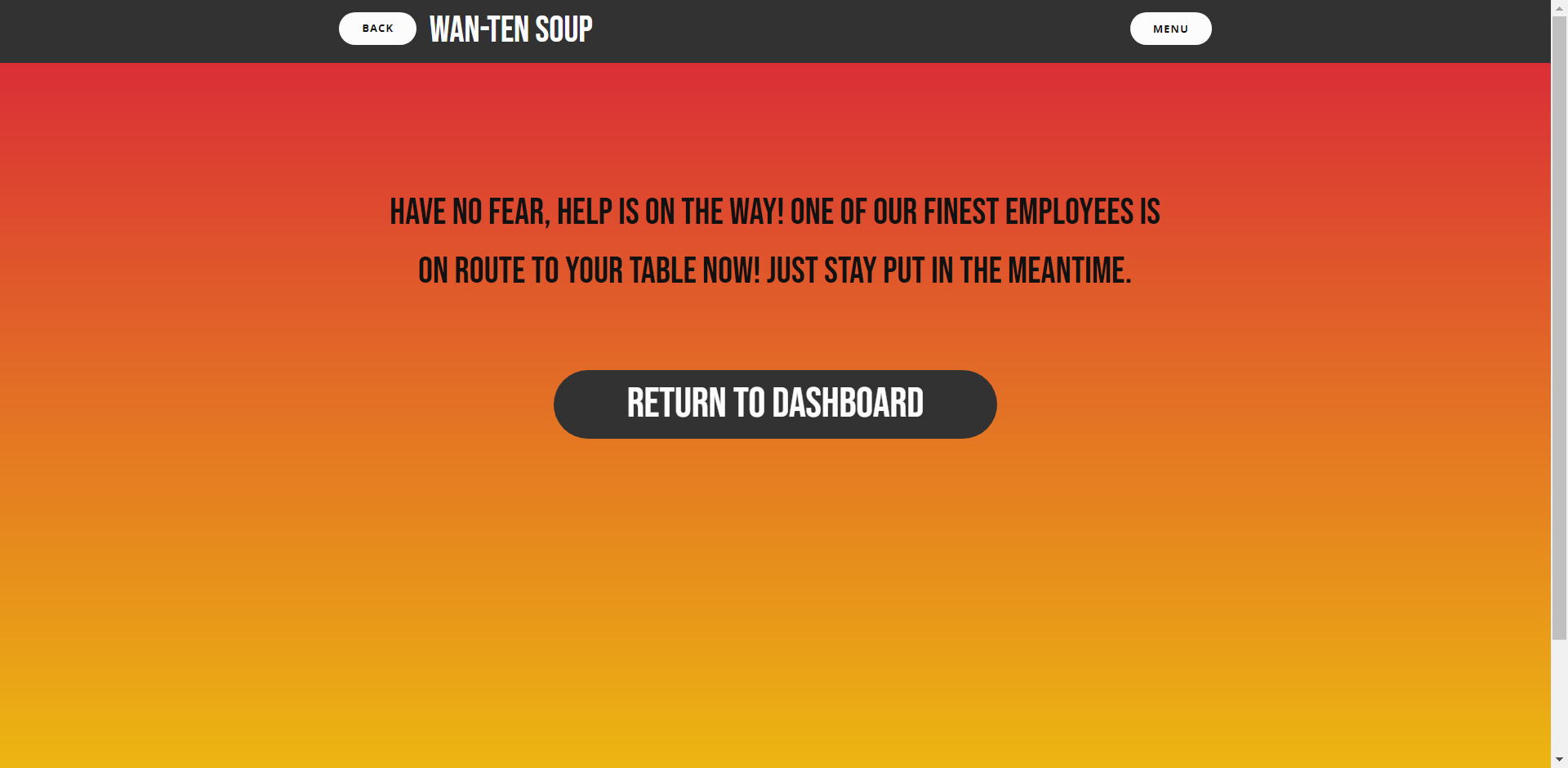


1. Customer C1 places a few orders while changing some of their quantities and deleting one from the order list. Customer C1 submits the order to be prepared and returns back to the customer dashboard.



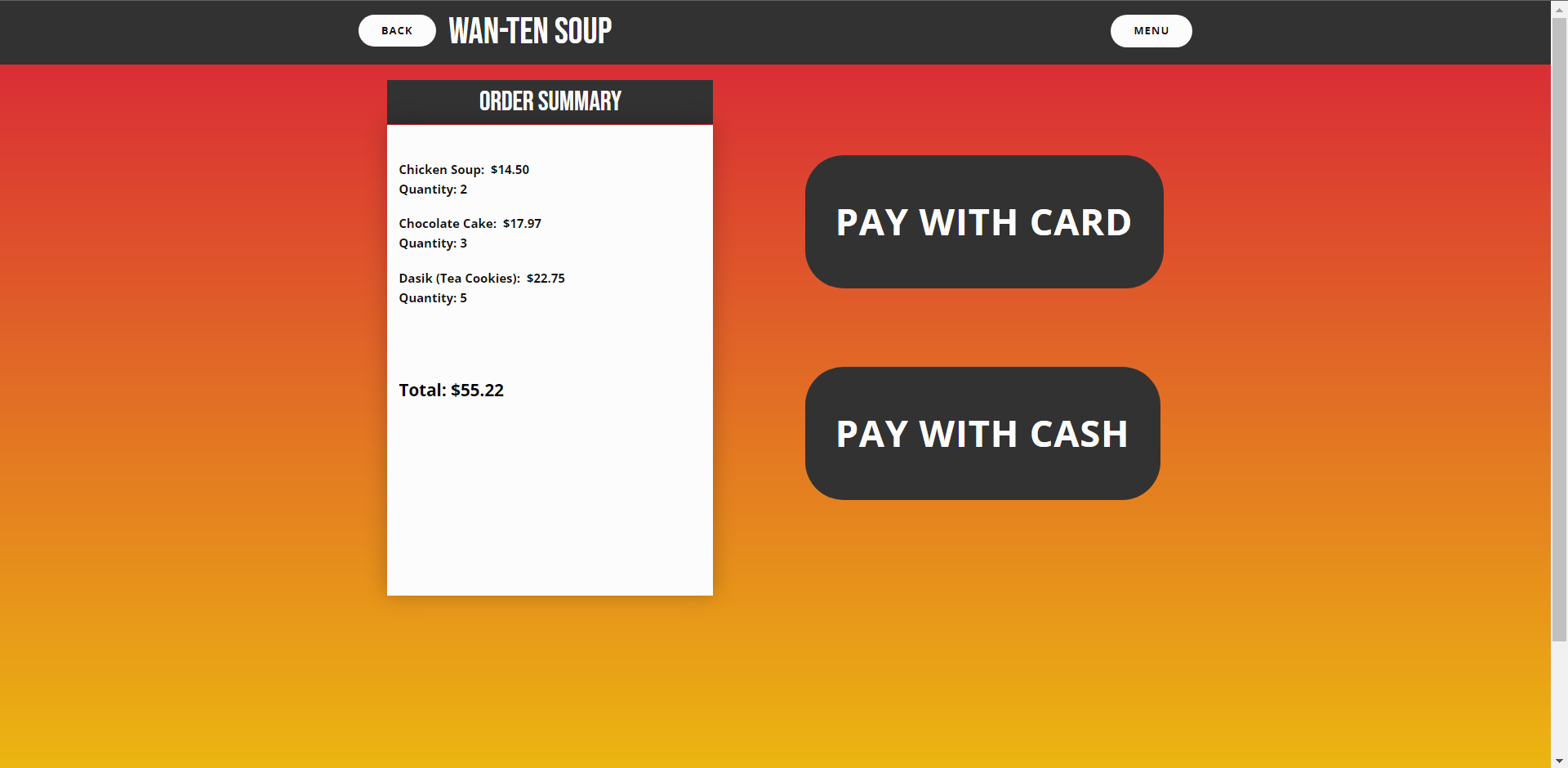
*Customer: Call For Service:*

1. Customer C1 utilizes the “call for service” button and returns back to the customer dashboard.



*Customer: Ready to Pay? Use case*

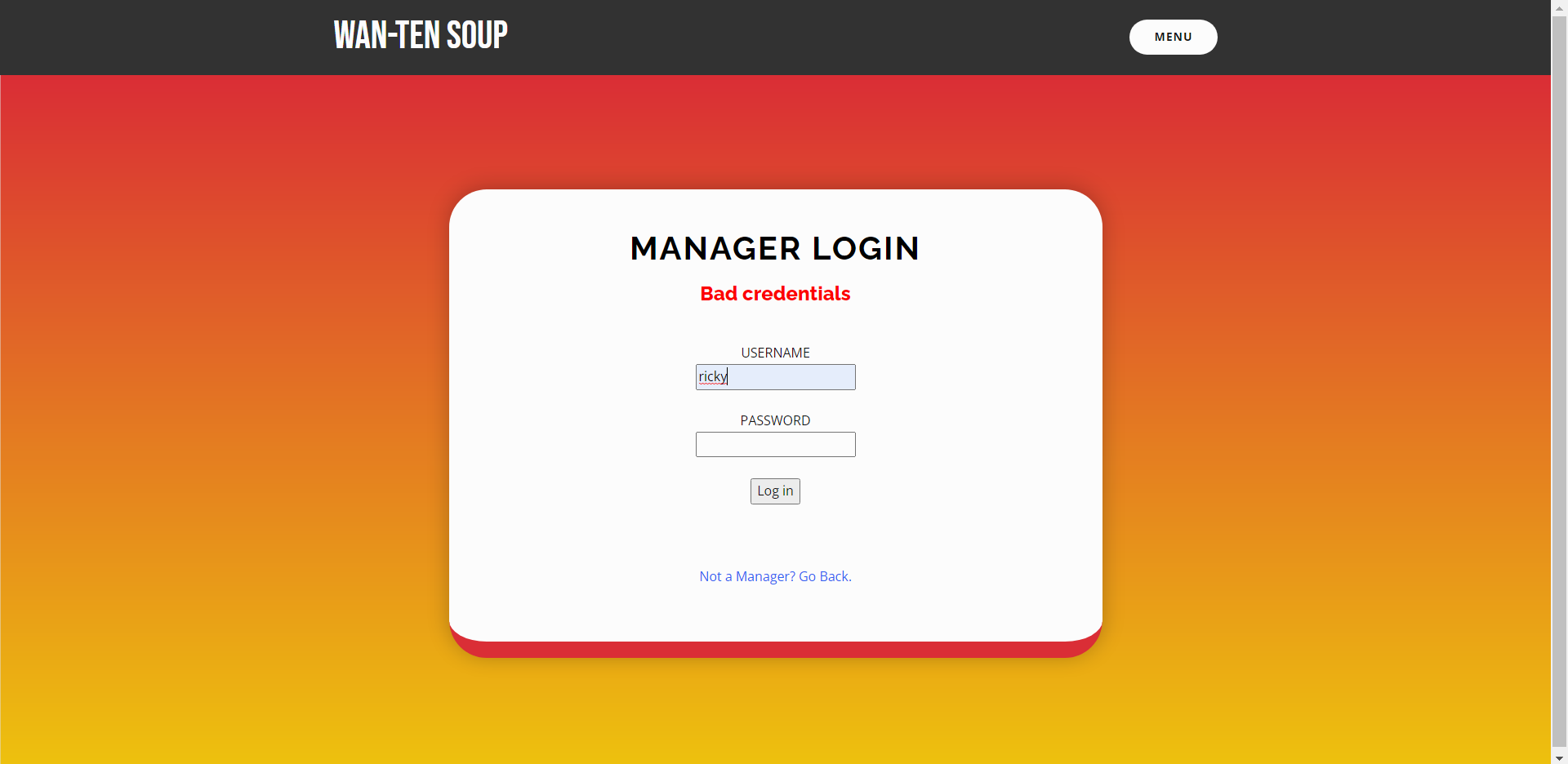
1. Customer C1 clicks on “Ready to Pay?” and views their order summary. Customer C1 pays with a card. Customer C1 exits.



1. Customer C2 enters and is logged in with a different table’s credentials. Customer C2 quickly places an order and then attempts to pay with cash. Customer C2 exits.

*Manager: Login use case:*

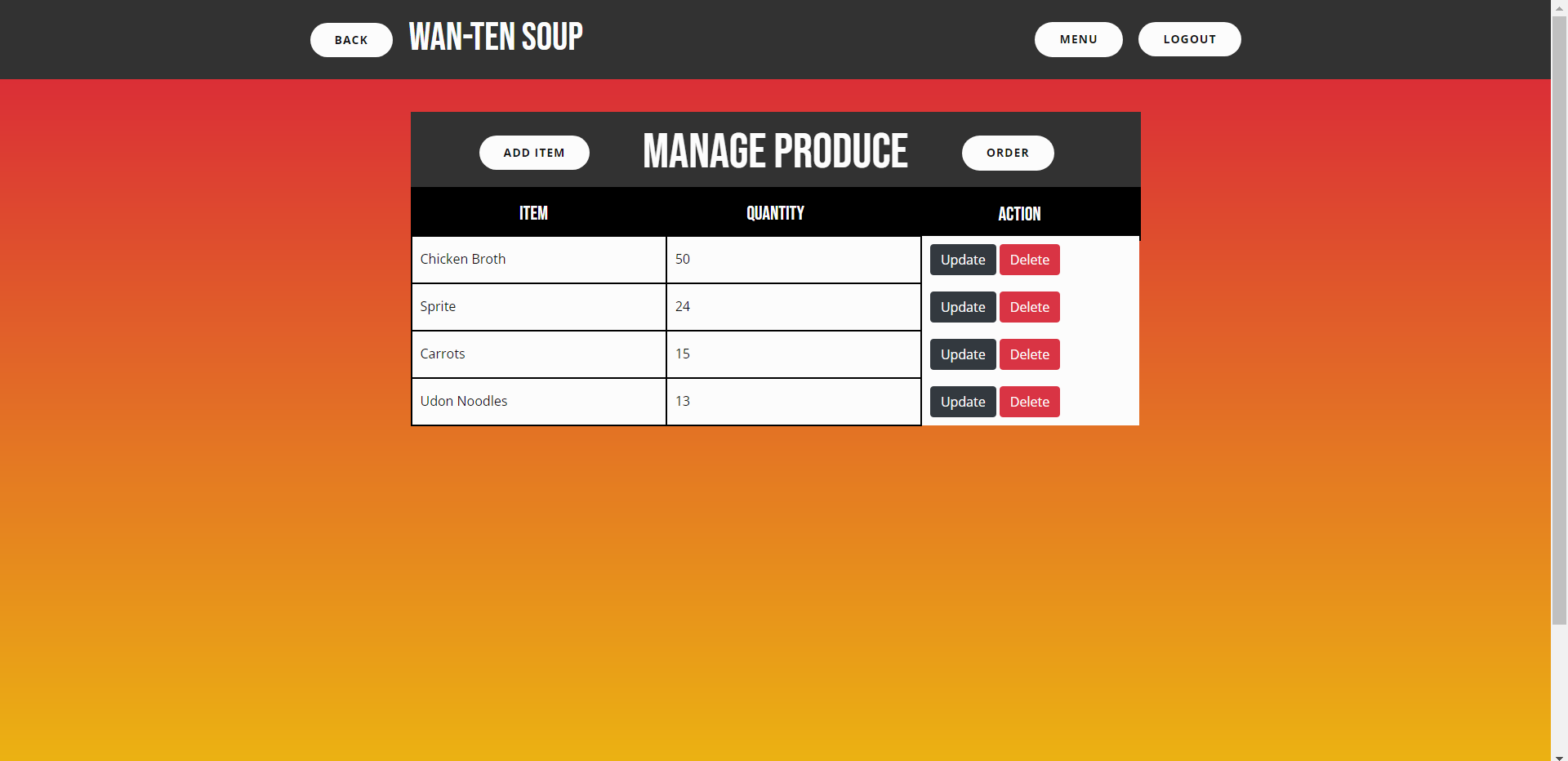
1. Manager M1 tries to log in for the first time but fails.



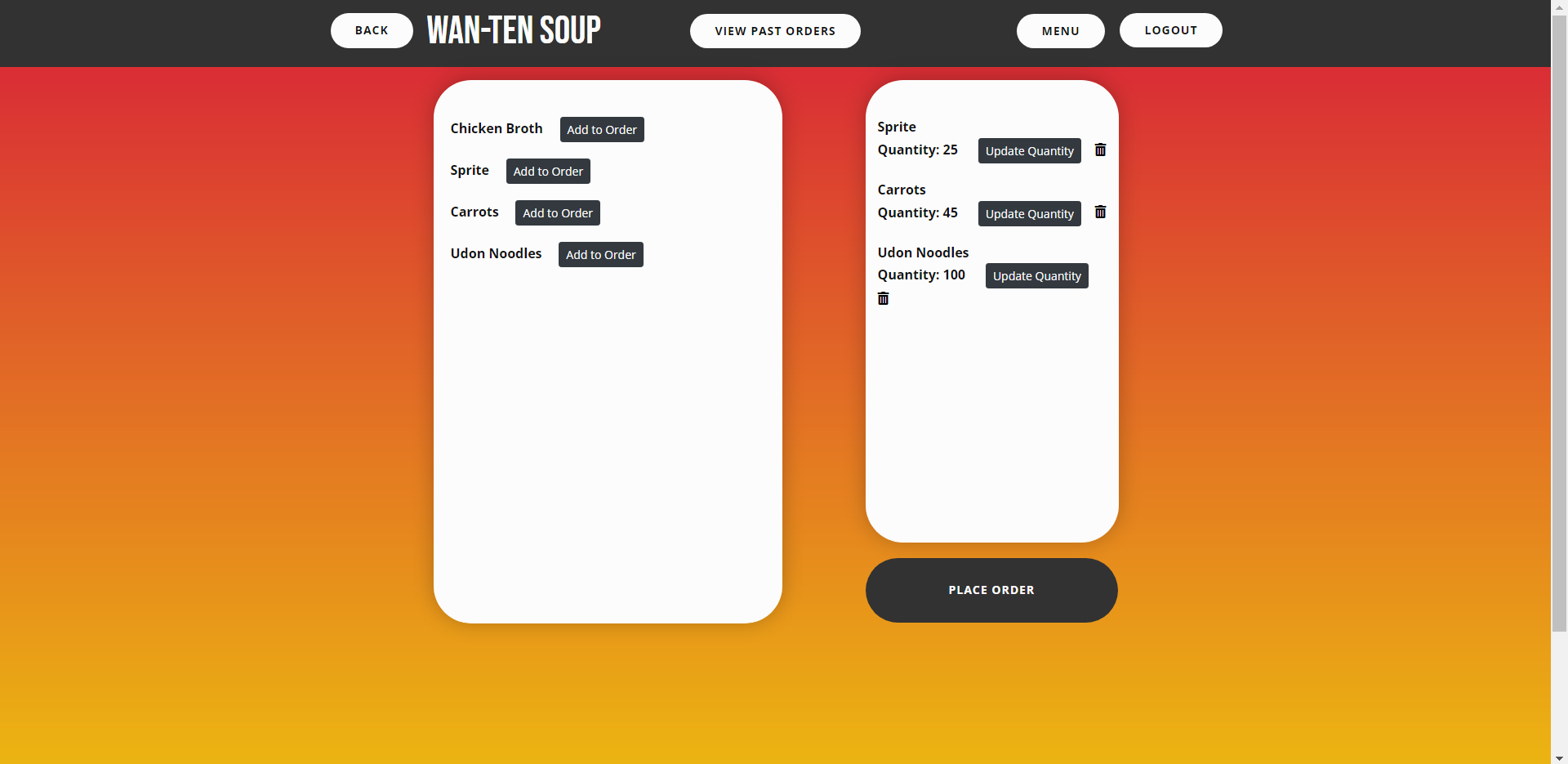
1. Manager M1 tries to log in again and succeeds. Manager M1 views the manager dashboard.

*Manager: Manage Produce use case:*

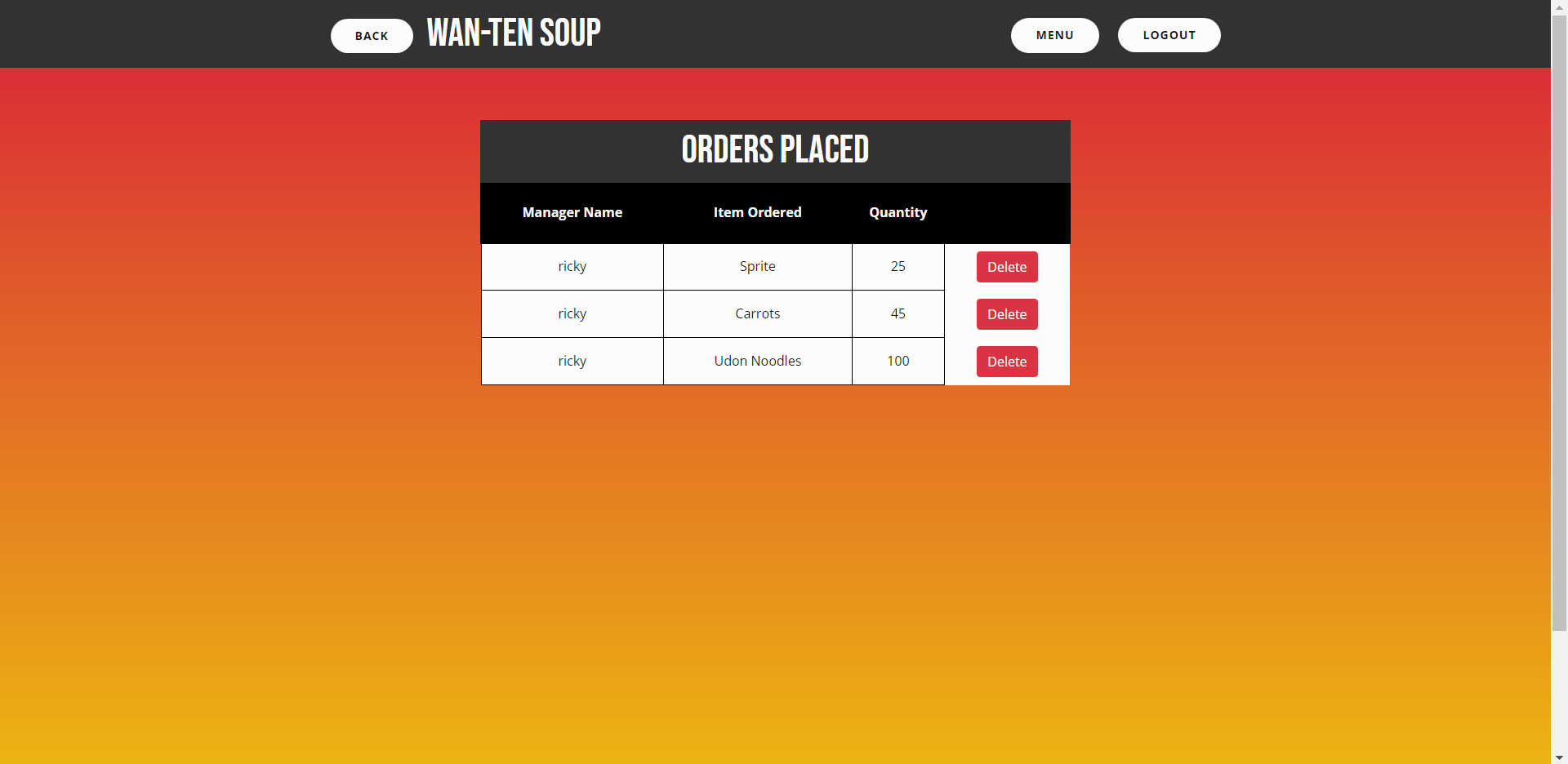
1. Manager M1 clicks on “Manage Produce” from the manager dashboard and views the list of items currently in stock.



1. Manager M1 updates an item and then deletes another item from the list.
2. Manager M1 clicks on “Add Item” and adds a new item to the list.
3. Manager M1 clicks on “Order” to order more items, changing the quantity and deleting items from the order list as well. M1 clicks on “Place Order” and the order is submitted. M1 is brought to an order confirmed page.



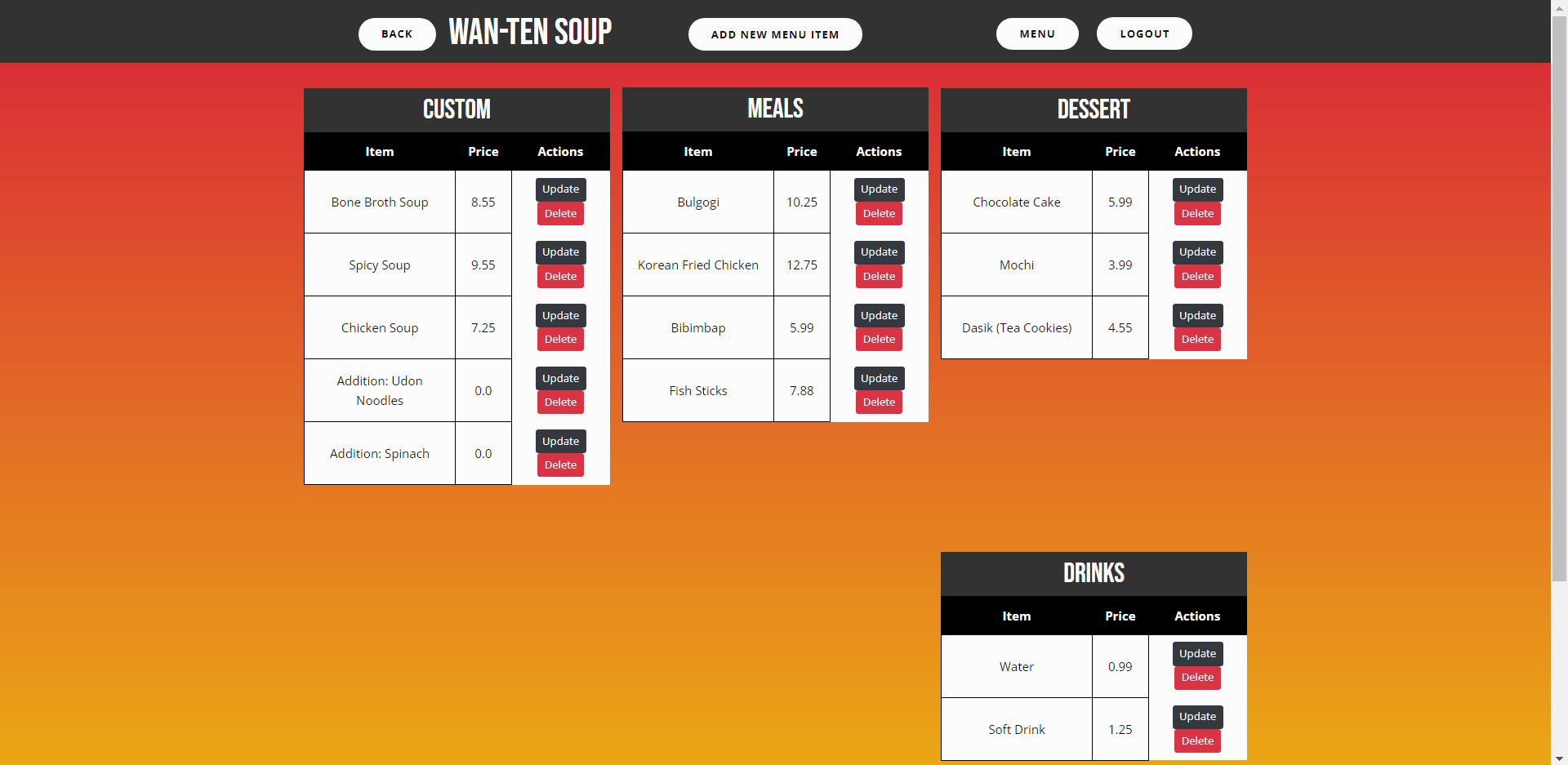
1. Manager M1 goes back to the “Manage Produce” list, clicks on “Order” again and then clicks on “View past orders” to see the order they just placed.



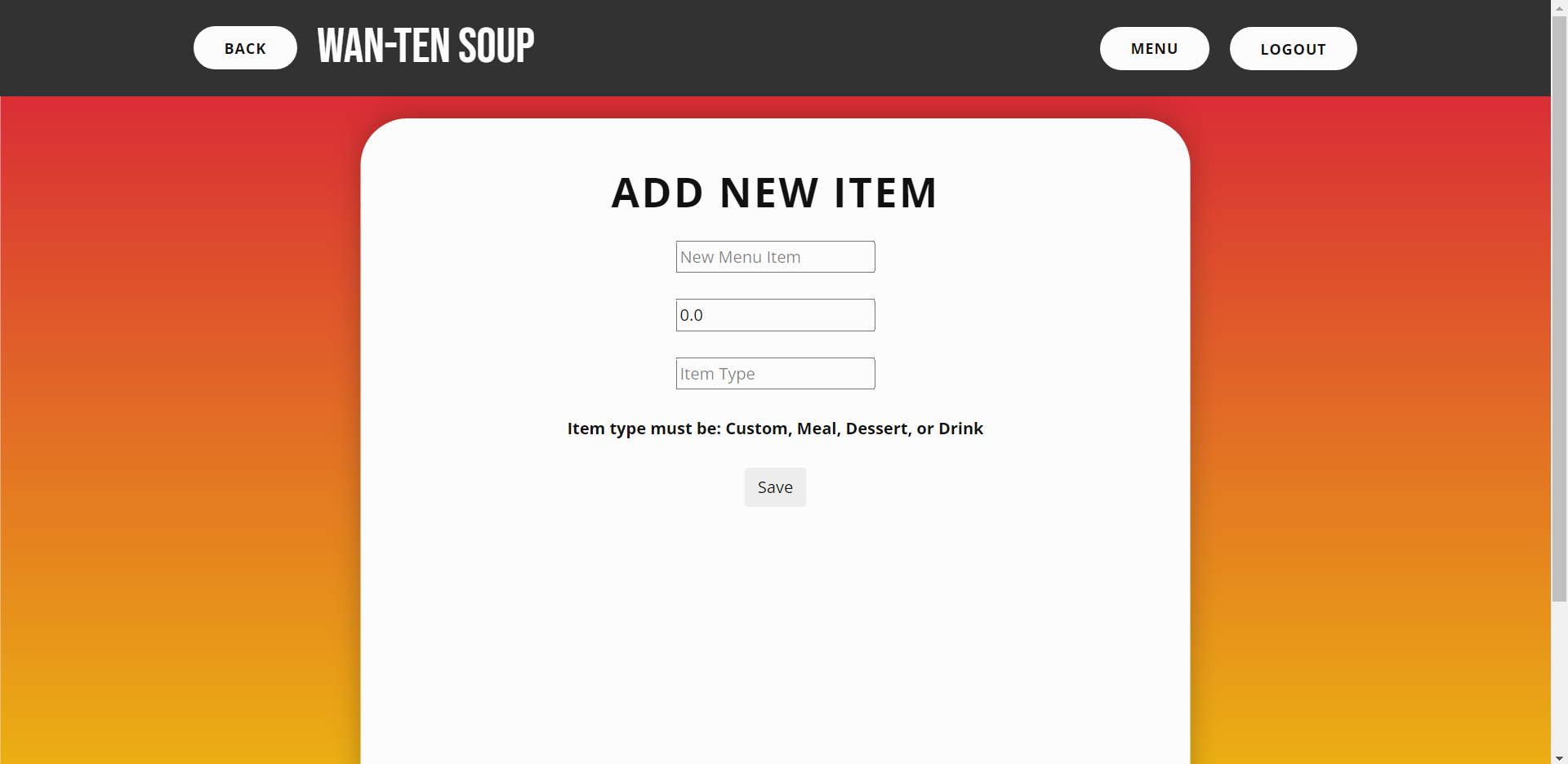
1. Manager M1 deletes that order from the list and then backtracks back to the manager dashboard.

*Manager: Manage Menu use case:*

1. Manager M1 clicks on “Manage Menu” from the manager dashboard and views the menu.



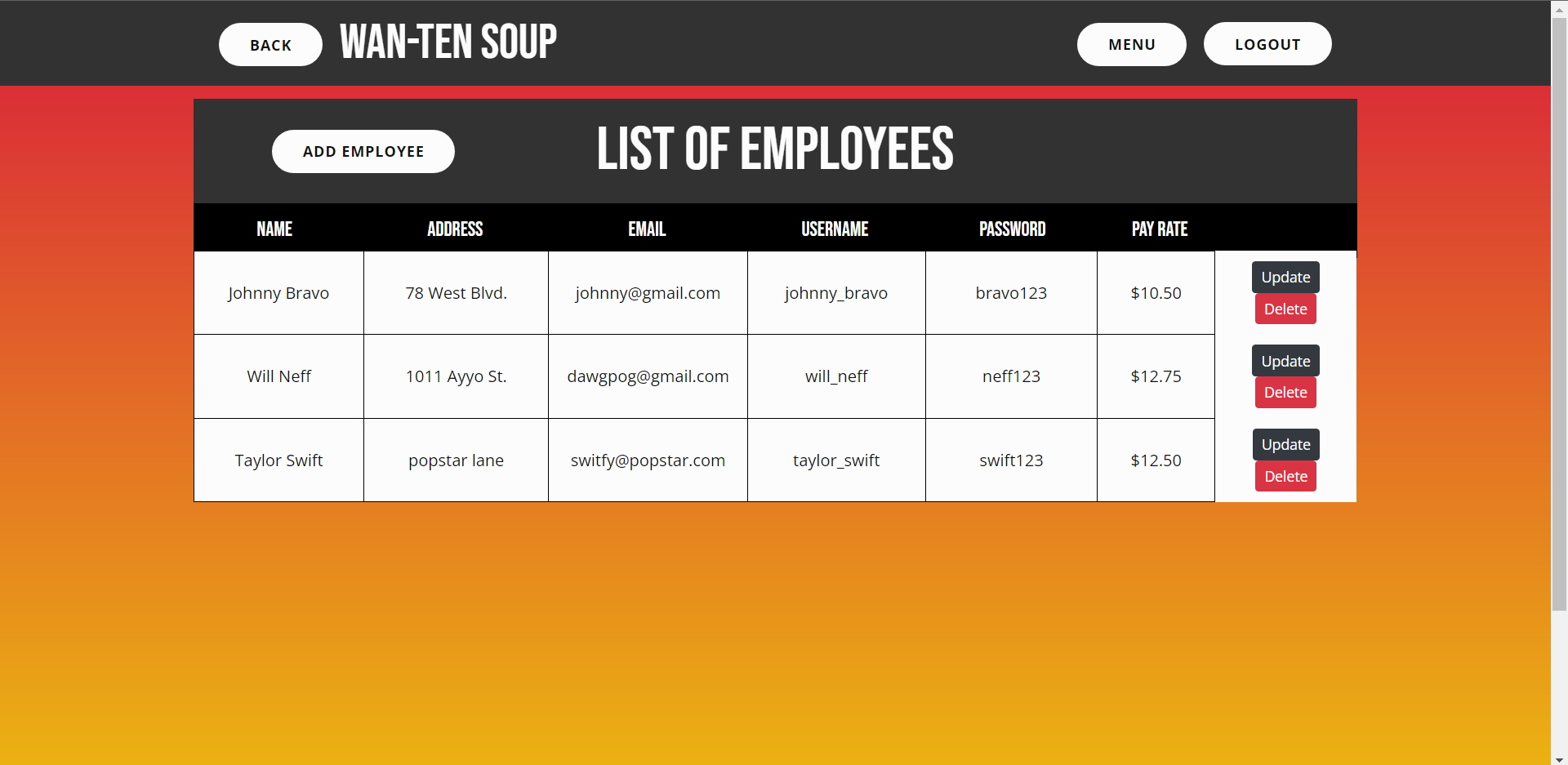
1. Manager M1 updates the quantity of any item on the menu and deletes an item from the menu as well.
2. Manager M1 clicks on “Add new menu item” and adds a new item to the menu.



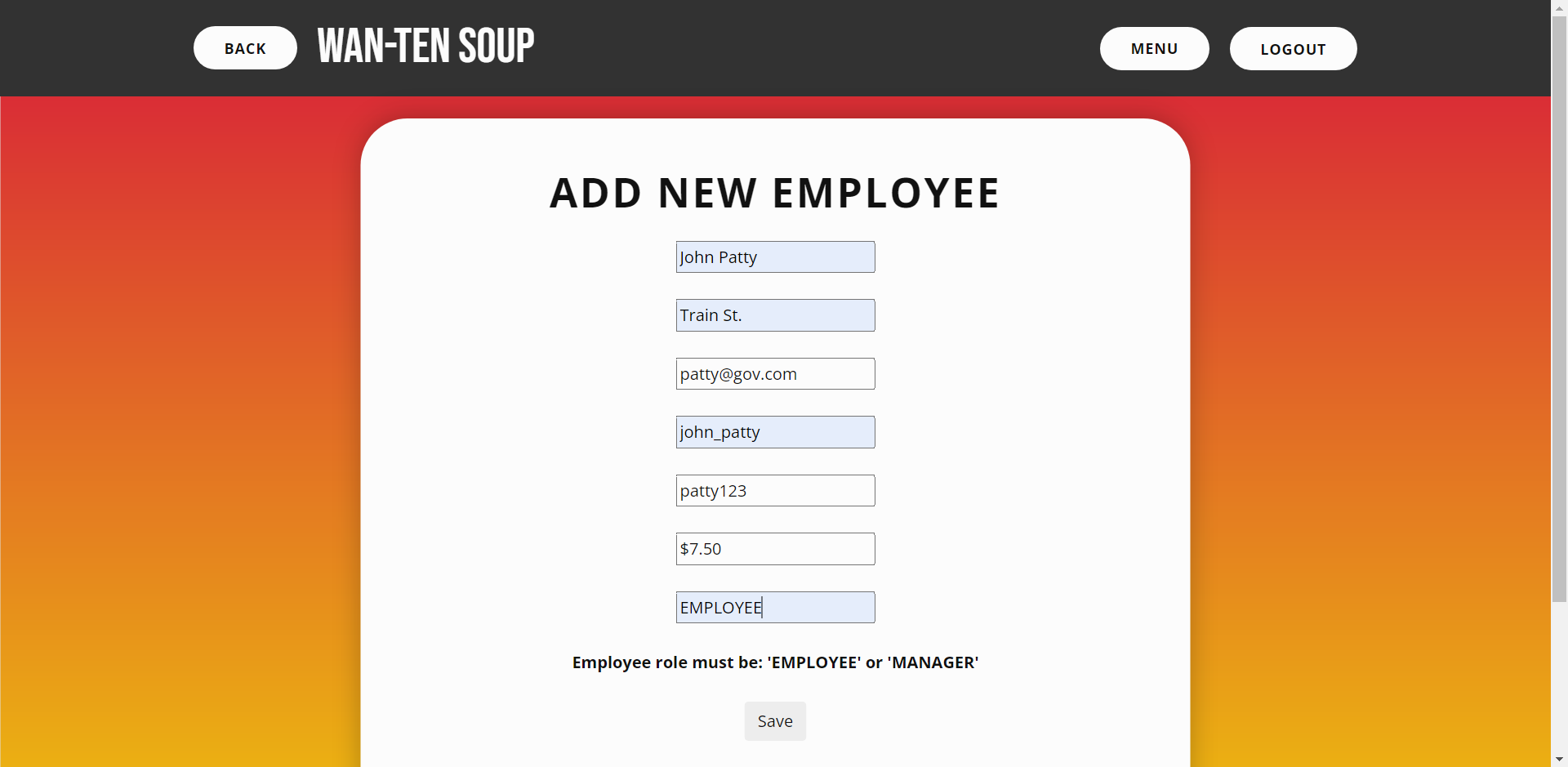
1. Manager M1 clicks on “Menu” to see that the new menu item was created and put on the menu for everyone to see. Manager M1 backtracks back to the manager dashboard.

*Manager: Manage Employees use case:*

1. Manager M1 clicks on “Manage Employees” from the manager dashboard and views a list of employees.



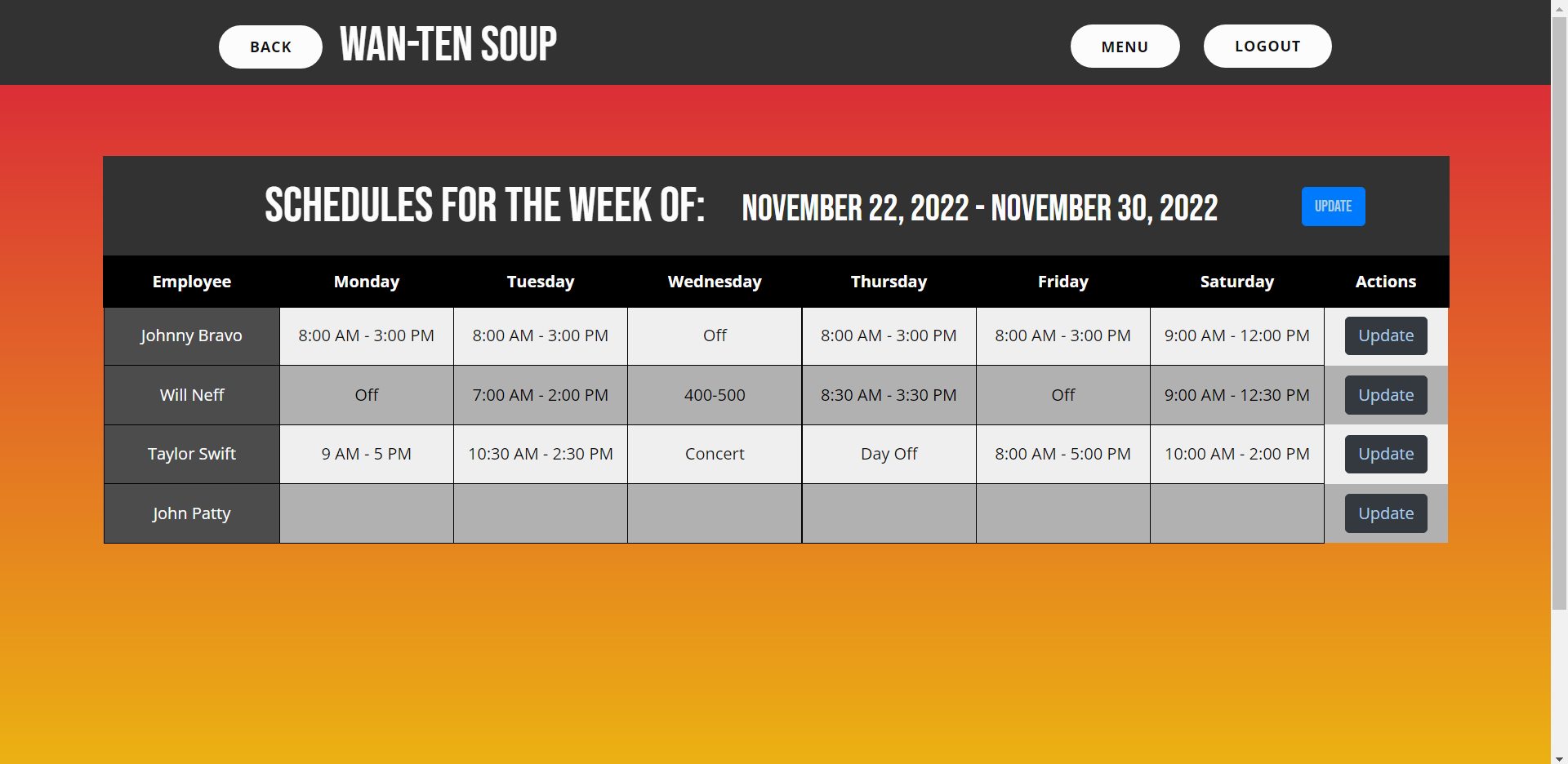
1. Manager M1 updates the information of one employee and then deletes another employee from the list.
2. Manager M1 clicks on “Add Employee” and adds employee E1 to the list of employees.



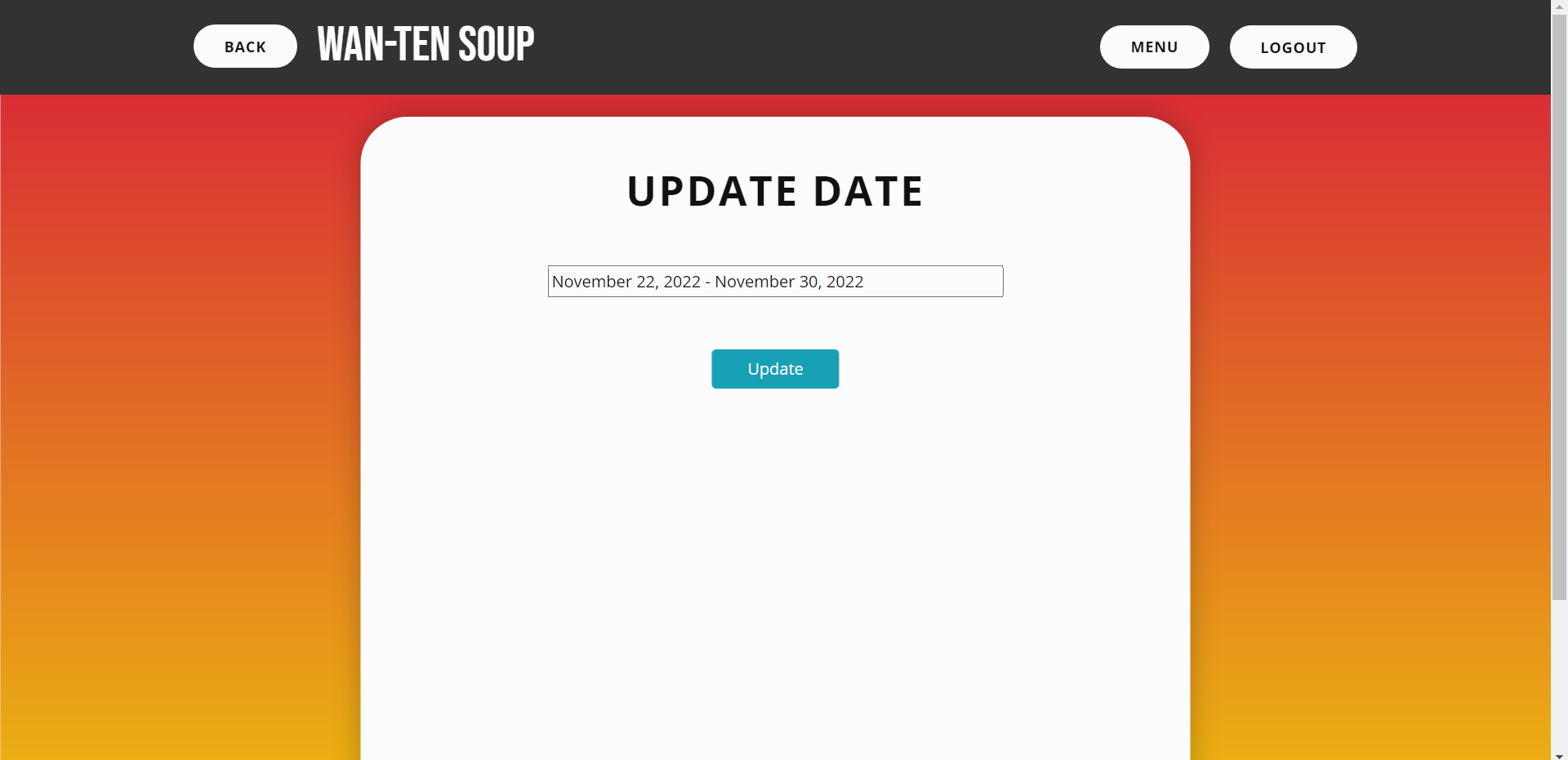
1. Manager M1 backtracks back to the manager dashboard.

*Manager: Modify Schedules use case:*

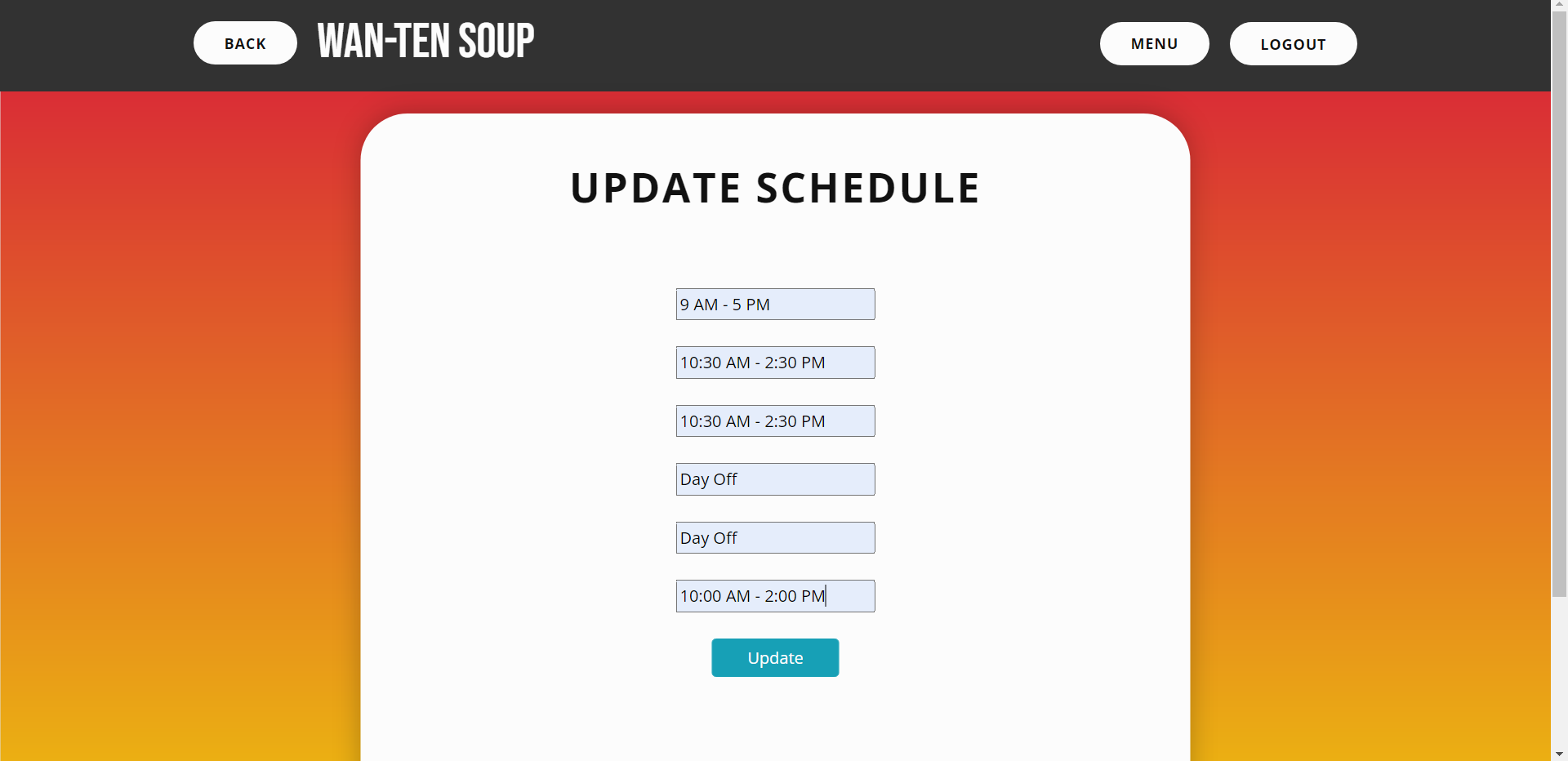
1. Manager M1 clicks on “Modify Schedules” from the manager dashboard and views a list of schedules for all employees.



1. Manager M1 changes and updates the weekly date for the schedules.



1. Manager M1 updates the schedule of one of the employees that doesn't have a blank schedule.
2. Manager M1 gives employee E1 a schedule for the week. Manager M1 backtracks back to the manager dashboard. Manager M1 exits.

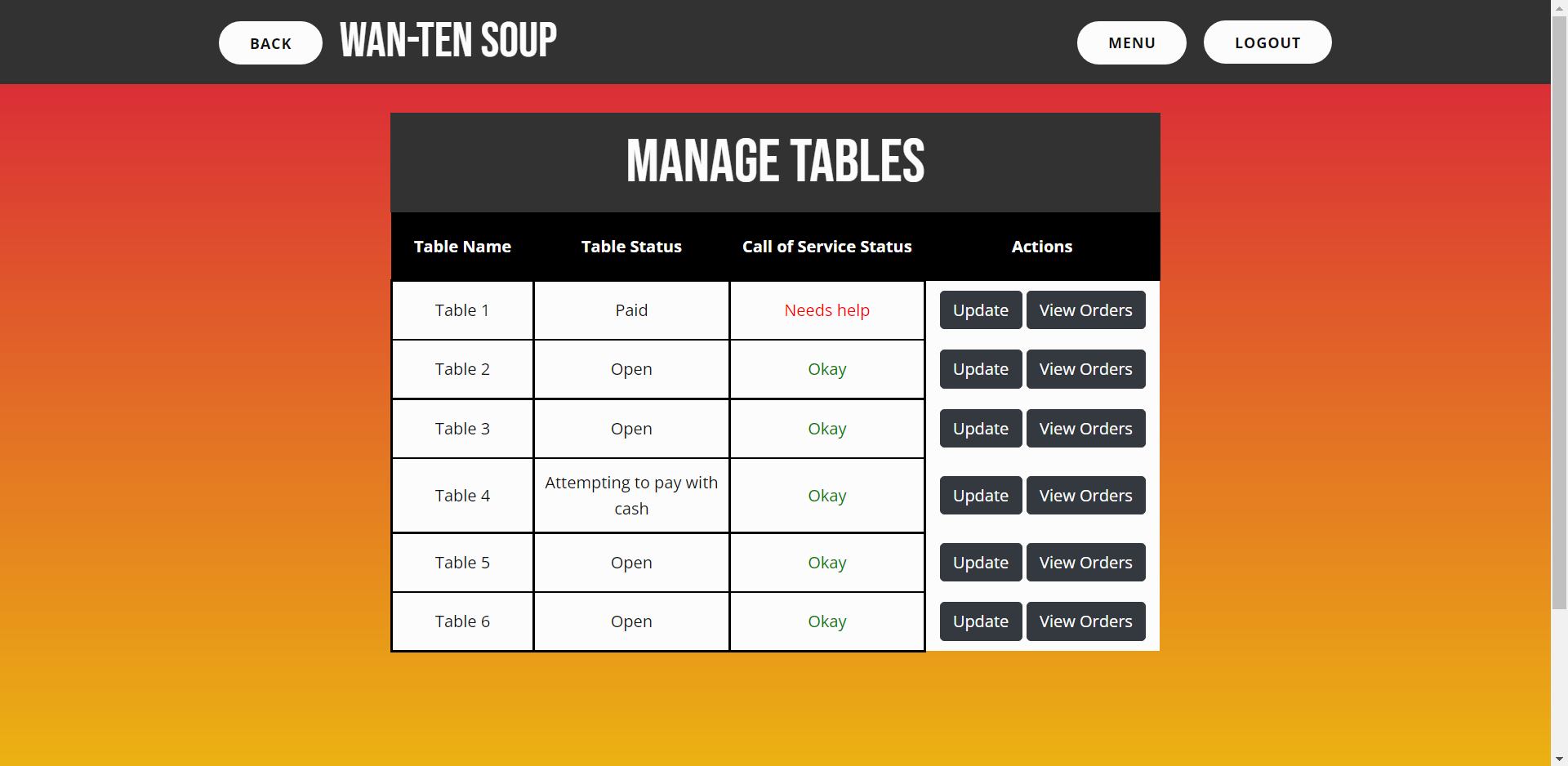


*Employee: Login use case*

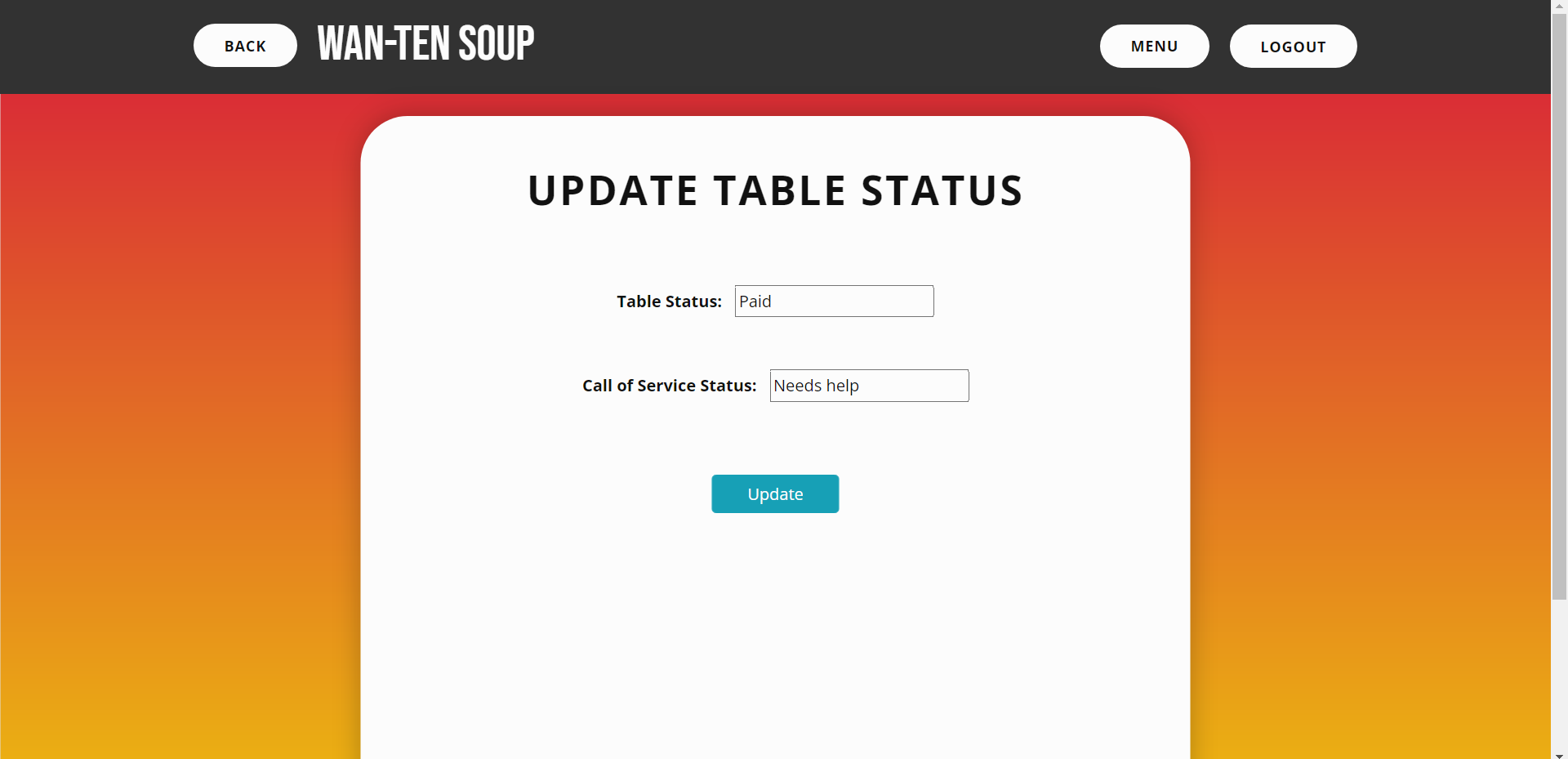
1. Employee E1 logins for the first time and views the employee dashboard.

*Employee: Manage tables and view orders use case*

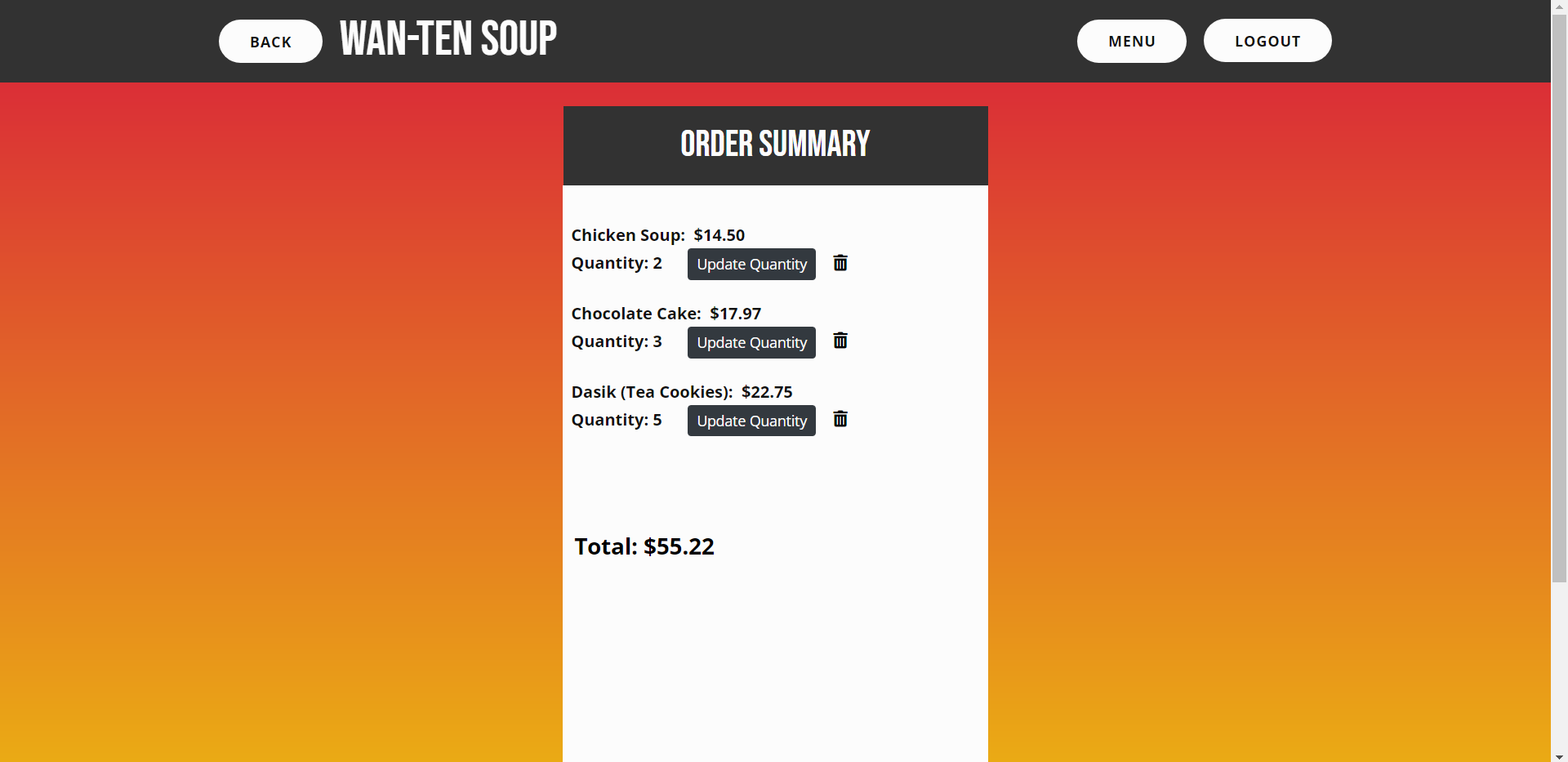
1. Employee E1 clicks on “Manage Tables and View Orders” and views the list of tables.



1. Employee E1 views the table's status and can update whenever a customer pays, leaves, or has ordered. Employee E1 also changes the status of the service call done by customer C1.

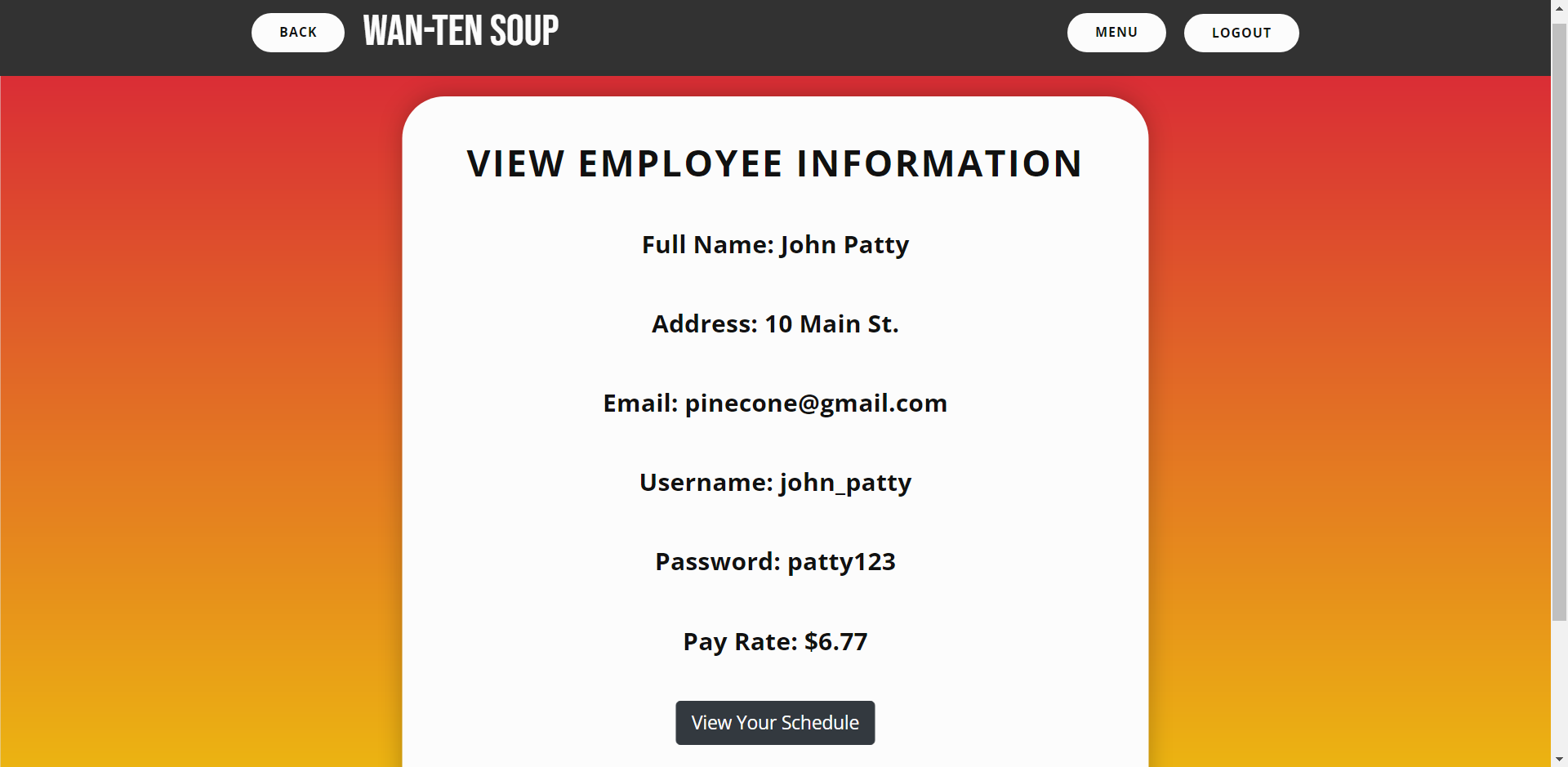


1. Employee E1 can click on “View Orders” for the table that customer C1 and C2 were at and view and edit their orders. Employee E1 backtracks back to the employee dashboard.



*Employee: View Employee Information use case*

1. Employee E1 clicks on “View Employee Information” and views their information.



1. Employee E1 clicks on “View Your Schedule” and views their schedule. Employee E1 backtracks back to the employee dashboard. Employee E1 exits.

