***Software Engineering***

***Software Requirements Specification***

***(SRS) Document***

**Restaurant Manager**

**September 20, 2022**

**Version 1.0**

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

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

**Table of Contents**

[**1. Introduction**](#_o85guvuqbuid) **3**

[1.1. Purpose](#_4klqc4ie4zml) 3

[1.2. Document Conventions](#_78563pjg5bch) 3

[1.3. Definitions, Acronyms, and Abbreviations](#_b7526micyjr4) 3

[1.4. Intended Audience](#_r9o2iey1yvsc) 3

[1.5. Project Scope](#_wy7aeyn8qcth) 4

[1.6. Technology Challenges](#_qtmb71pd7klo) 4

[1.7. References](#_xwjjfp8yqvof) 4

[**2. General Description**](#_tpo7govv27l4) **4**

[2.1. Product Perspective](#_7hczuow2j60s) 4

[2.2. Product Features](#_x21kt6p69cap) 4

[2.3. User Class and Characteristics](#_4mi7doi28xt5) 5

[2.4. Operating Environment](#_nj33jgta5p6h) 5

[2.5. Constraints](#_9y3b17j0fmft) 5

[2.6. Assumptions and Dependencies](#_posjpt8qhvt4) 5

[**3. Functional Requirements**](#_f7jvrqly81xd) **5**

[3.1. Primary](#_ru4m2xa2t8dw) 5

[3.2. Secondary](#_uguxodb4pxjk) 5

[3.3. Use Cases](#_d9mo1jdgh6pl) 6

[3.3.1. Actor: Manager (Kristin Cattell)](#_cjum2w46pa7i) 6

[3.3.2. Actor: Employee (Atsoupe Bessou)](#_axxmwgsx3s7l) 6

[3.3.3. Actor: Customer (Ewan Allen)](#_okwbxl7pfm6t) 7

[**4. Technical Requirements**](#_3w1m4jdn8z4i) **8**

[4.1. Operating System and Compatibility](#_r59hljnlse56) 8

[4.2. Interface Requirements](#_v1o3kgtzanpw) 8

[4.2.1. User Interfaces](#_hwfs9f1bil2r) 8

[4.2.2. Hardware Interfaces](#_c994zanji1dt) 8

[4.2.3. Communications Interfaces](#_j7qorheaob3z) 8

[4.2.4. Software Interfaces](#_e6stgtlhrmqu) 8

[**5. Non-Functional Requirements**](#_3h6suql7k34a) **8**

[5.1. Performance Requirements](#_i1vhgy96eg2l) 8

[5.2. Safety Requirements](#_d5f0vsc0fxm9) 9

[5.3. Security Requirements](#_1l7e05i3z49v) 9

[5.4. Software Quality Attributes](#_2jmv5pvw8iz) 9

[5.4.1. Availability](#_m6zxd79se6l0) 9

[5.4.2. Correctness](#_o3tnteojoger) 9

[5.4.3. Maintainability](#_9wryzefh7fdm) 9

[5.4.4. Reusability](#_yb0lfdi2zqz4) 9

[5.4.5. Portability](#_mv3czbzncdm) 9

[5.5. Process Requirements](#_su70olcav0rf) 10

[5.5.1. Development Process Used](#_e8hda2bpiry0) 10

[5.5.2. Time Constraints](#_bq2ke8hd8c0l) 10

[5.5.3. Cost and Delivery Date](#_xx4rcrys6rq2) 10

[5.6. Other Requirements](#_zcaectccywls) 10

[5.7. Use-Case Model Diagram](#_qa3gnnifeqwe) 10

[5.8. Use-Case Model Descriptions](#_4uithxblm2e6) 10

[5.8.1. Actor: Manager (Kristin Cattell)](#_392bpaxg136c) 10

[5.8.2. Actor: Employee (Atsoupe Bessou)](#_4fbdiss02clj) 11

[5.8.3. Actor: Customer (Ewan Allen)](#_1c3vqey7kojq) 11

[5.9. Use-Case Model Scenarios](#_53z4u8ya1qde) 11

[5.9.1. Actor: Manager (Kristin Cattell)](#_v70zpwcd1d55) 11

[5.9.2. Actor: Employee (Atsoupe Bessou)](#_babd2j39im) 13

[5.9.3. Actor: Customer (Ewan Allen)](#_1h5xs2akkrcs) 14

[**6. Design Documents**](#_lib1zbjznpca) **16**

[6.1. Software Architecture](#_9mb3baornyuf) 16

[6.1.1. High-Level Database Schema](#_9m1iqsj5z9hp) 18

[6.2. Software Design](#_z9ya4tc4jklb) 20

[6.2.1. State Machine Diagram: Manager (Kristin Cattell)](#_7c517qr645w9) 20

[6.2.2. State Machine Diagram: Employee (Atsoupe Bessou)](#_t8phyt9iufb3) 21

[6.2.3. State Machine Diagram: Customer (Ewan Allen)](#_agp3pqy9eddy) 22

[6.3. UML Class Diagram](#_akqhbiryg4s6) 23

[6.3.1. UML Class Diagram: Manager](#_mly0zmnr9f10) 23

[6.3.2. UML Class Diagram: Employee](#_vm9pon7ztyb2) 24

[6.3.3. UML Class Diagram: Customer](#_o9e6s5uezamq) 24

[6.3.4. UML Class Diagram: Others (Item, User, HomeController, SecurityConfigs)](#_i7ms77b4bvhe) 25

# 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 soon-to-be-determined function within the software. |

## 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 an API that will display the date and time at any given moment within the dashboard of all customers, employees, and managers.

# 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 customers to request for their receipt from the interface.
* 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 and managers.
* Protection scheme so that customers can only alter and see their orders and not other customers’ orders.
* The application will utilize an API to display the date and time on the dashboard for all customers, employees, and managers.

## 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 adds or modifies time slots for specific selected dates. |
| Basic Flow: | 1. The use case starts when a user indicates that they want to manage and modify an employee’s schedule. 2. The system displays a list of employees that the restaurant currently employs. 3. The user selects which employee that they want to manage and modify the schedule of. 4. The system displays a calendar of the current month. When clicking on a specific date, the system requests the time slot for the selected date. 5. The user enters the day and time slot to schedule an employee. 6. The system stores any new changes to an employee's schedule. |
| Alternative Flows: | Step 4: If there’s already a time slot occupying a selected date, then the system allows the user to modify or delete 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 layout of the restaurant and indicates which tables are empty or non-empty. 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 layout. |
| 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 delete that occupied table slot and create it to be filled with the new, arriving customers. |

| 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 layout of the restaurant and indicates which tables are empty or non-empty. 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, indicating which orders have already been fulfilled. 5. The user applies changes to orders as to whether they’ve been fulfilled, not completed, or canceled. 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 set the quantity of the product to “0” 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 displays a selection of what services are provided to the user. 3. The user selects the specific service that they need. 4. 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. Future consideration will be given to menus and other things that may vary case-to-case.

### Hardware Interfaces

* For now, just any computer capable of running Java is all the project requires. If need be to switch to a more accessible means then we will update as soon as possible.
* 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 an API.

### Communications Interfaces

* It must be able to connect to the internet as well as a potential database or text file.
* The communication protocol, HTTP, must be able to connect with an API and return information.

### Software Interfaces

* SpringBoot will interact with the Java project and HTML file 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.

# 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.

## 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.

## 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 layout 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 changing the quantity to “0” for that product.
* If the manager makes a mistake on scheduling employees, they can easily go and alter an employee’s time slot.
* 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

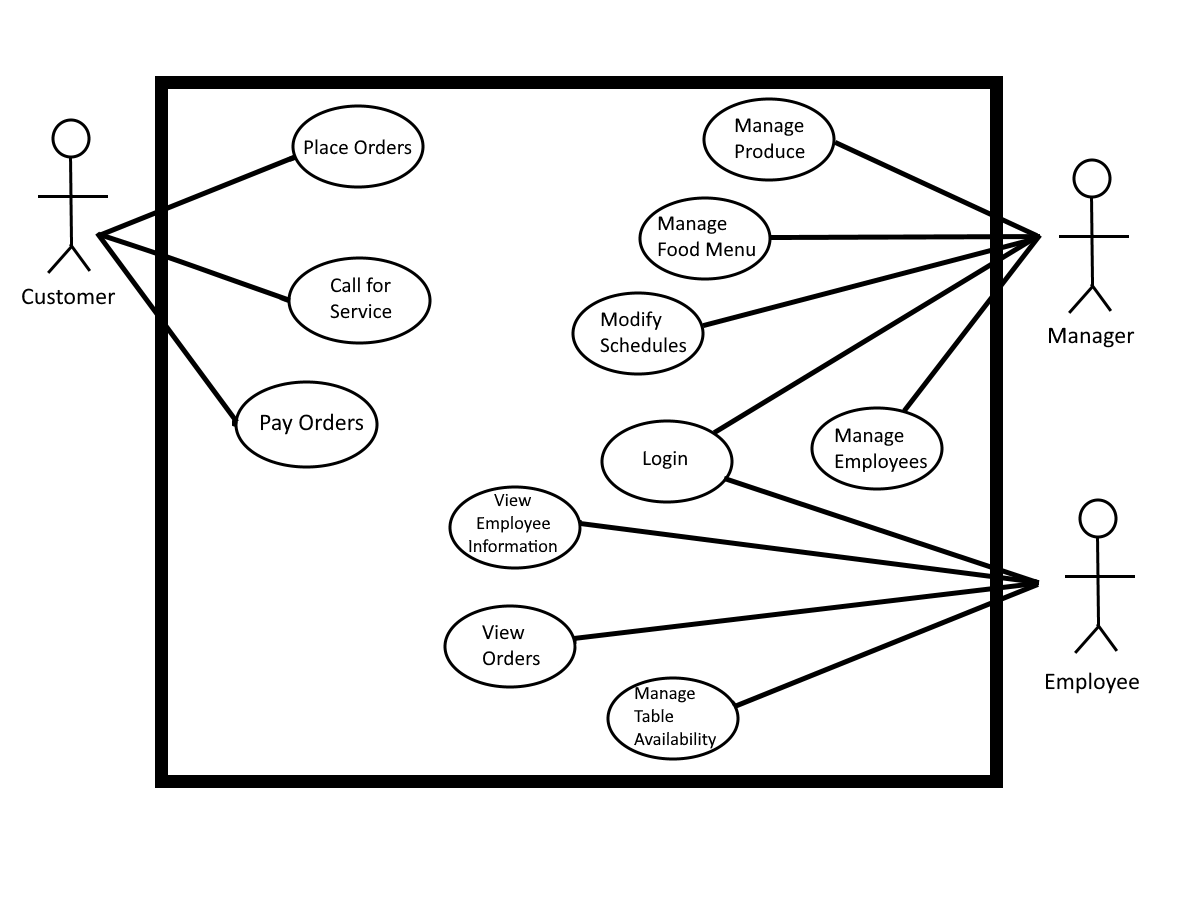
* The design document for the application is to be completed by October 4, 2022.
* The prototype of the application is to be completed by October 18, 2022.

### 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 adding or deleting time slots on particular days. The manager can also modify existing time slots by clicking on any time slot and changing the times of it. The time slots would be updated and the changes would also be displayed to the employees to indicate that the manager has made changes to an employee’s schedule.
* **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, check for availability of products, and be able to make changes to orders when some items are out of stock. The employee can also issue a refund when necessary.
* **Manage Table Availability**: The employee can access and view what tables are taken.
* **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, as well as leave comments for the cooking staff on specifics for their food. 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 a member of the waiting staff or a manager, depending on the nature of their servicing needs. 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. The manager should be able to request a password reset.
  + **Other Activities**: The manager can reset their password by using the “Forgot Password” link.
  + **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 has made a mistake on the request to the vendor. The manager should be able to cancel the order within a certain amount of time.
  + **Other Activities**: The manager can cancel any order they make to a vendor within 45 minutes of initially sending the request by hitting Cancel next to the order request.
  + **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 system will tell the manager about the duplication of food/drink items and the manager can decide which one to keep and which one to delete on 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 on their dashboard in the Food Menu tab.
* **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 add/delete any time slots on any given day on the displayed calendar.
  + **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 using the Edit button that’s placed in every time slot so that 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 or view, being able to edit an employee’s information or pay rate if need be.
  + **What Can Go Wrong**: The manager can accidentally delete an employee from the database.
  + **Other Activities**: The manager can hit the Undo button after they have deleted an employee.
  + **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. The employee should be able to reset it.
  + **Other Activities**: The employee should be able to reset their password by using the “forgot password” link.
  + **System State on Completion**: The employee can log in the system and view the dashboard and can check the items that are available and out of stock and the menu.
* **View Orders**:
  + **Initial Assumption**: The employee has logged into their account and can view their dashboard and navigate to the 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 using a status drop-down box to select the correct status of the order.
  + **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 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 using a status drop-down box 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 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 by clicking on the Request Change button within the View Employee Information tab.
  + **System State on Completion**: An employee has successfully viewed their information, whether it be general information, their schedule, or their pay rate. If a change is requested, the system will display a Completed Request message once the request is submitted.

### 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 or wants to delete a selection from 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 delete button 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 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 is given a selection of services and the customer selects the specific service they want, whether it be general menu help, food complaints, or help with navigating the system.
  + **What Can Go Wrong**: The customer requests for the wrong service or resolves their problem before an employee can come and assist them.
  + **Other Activities**: The customer can cancel their request of service by clicking on the Cancel button within the Call for Service tab.
  + **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 their dashboards.
* **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 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 using the Split Orders button so that different items are paid for with different payment options.
  + **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 tab.

# 

# Design Documents

## Software Architecture

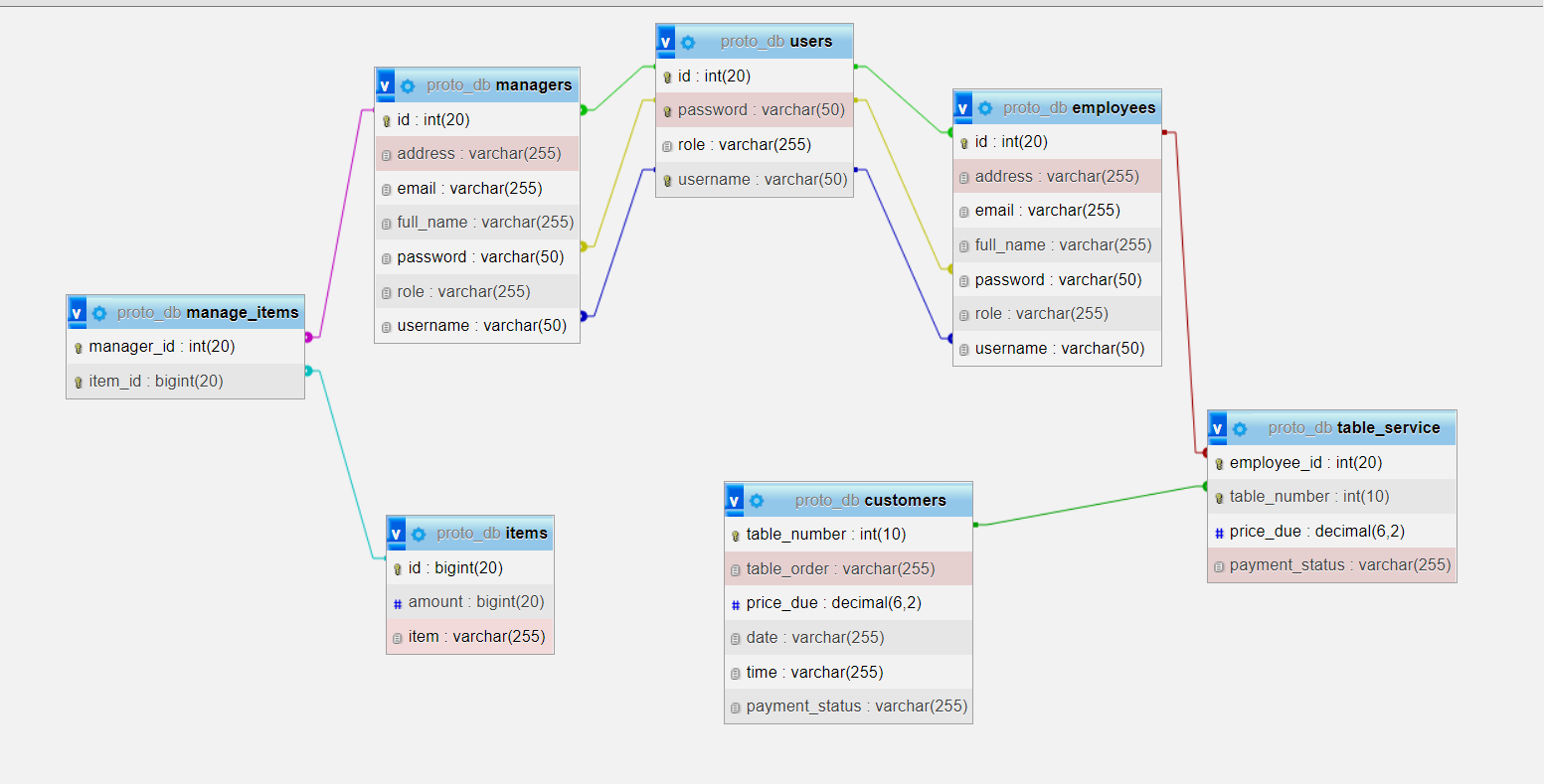
## 

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.
   * **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 database) 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 database) to access web pages that contain “/manager/” in the URL.
2. **Controllers**
   * **ItemController:** The class that contains all the HTTP request methods pertaining to displaying and modifying the table “items” in the database, which are the items that the restaurant has in the inventory (modifying schedules, managing produce, etc).
   * **ManagerController:** The class that contains all the HTTP request methods pertaining to displaying data from the database accessible to managers and allowing the manager to do a variety of actions.
   * **EmployeeController:** The class that contains all the HTTP request methods pertaining to displaying data from the database 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 database 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.
3. **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 database.
     + **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 database.
     + **ItemRepository:** The class that’s used by the item service layer/classes to access the database.
   * **Manager Model:** The “Manager” class creates the table “Managers” with the attributes that each entity will have.
     + **ManagerService:** Interface class that contains abstract methods that interact with and modify the data stored in the “Managers” table in the database. Will also have abstract methods to modify employee schedules/information and the menu.
     + **ManagerServiceImpl:** The class that implements the ManagerService class and overrides all the abstract methods to create functional methods to interact with and modify data stored in the “Managers” table. Will also override methods that will modify employee schedules/information and the menu.
     + **ManagerRepository:** The class that’s used by the manager service layer/classes to access the database.
   * **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/classes to access the database.
   * **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 database. 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/classes to access the database.
   * **Customer Model:** The “Customer” class creates the table “Customers” with the attributes that each entity will have.
     + **CustomerService:** Interface class that contains abstract methods that interact with and modify the data stored in the “Customers” table in the database. Will also have abstract methods for actions done by the customer.
     + **CustomerServiceImpl:** The class that implements the CustomerService class and overrides all the abstract methods to create functional methods to interact and modify the data stored in the “Customers” table. Will also override methods that implement actions done by the customer.
     + **CustomerRepository:** The class that’s used by the customer service layer/classes to access the database.
   * **Assumptions:**
     + Assume that one or multiple controllers in the high-level architecture diagram will call an API and that API will retrieve some data and certain web pages will display that data.

## High-Level Database Schema

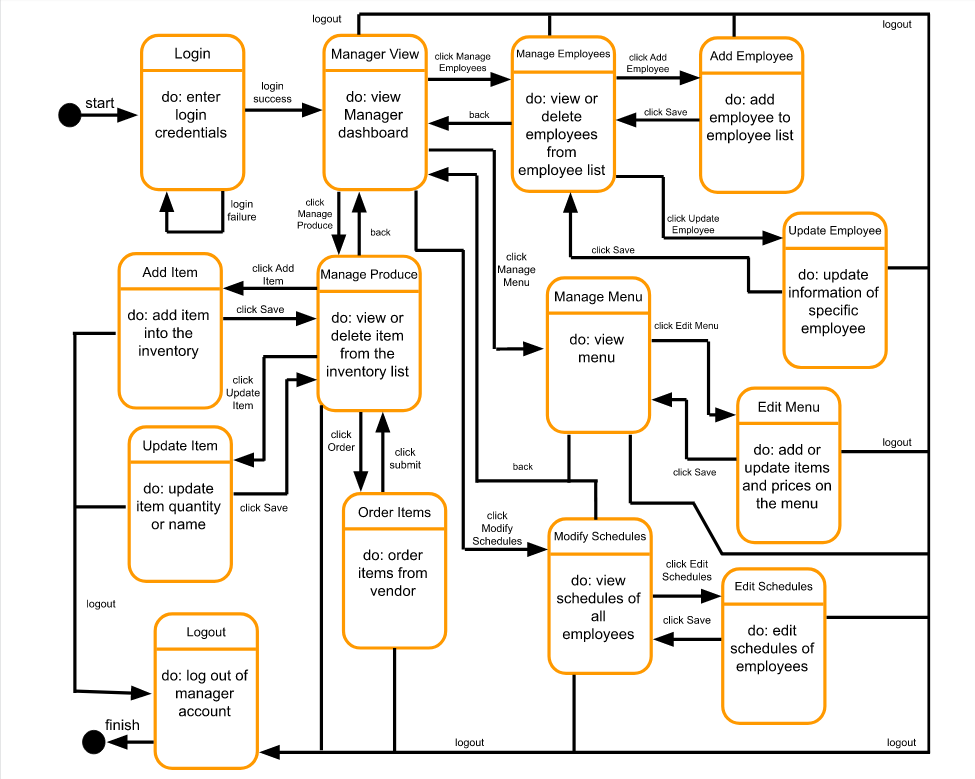
* **“items” Table:** Will store the item’s ID, item’s name, and the quantity of that item that’s currently in stock in the restaurant’s inventory.
* **“Managers” Table:** Will store the manager’s ID, full name, street address, email, username, password, and role.
* **“Users” Table:** Will store every user's (managers and employees) ID, username, password, and role.
* **“Employees” Table:** Will store every employee’s ID, full name, street address, email, username, password, and role.
* **“Customers” Table:** Will store every table’s table number, order, price of order, status of payment, and date/time that the table was used.
* **“table\_service” Table:** Table that shows the relationship between a table and the employee that’s serving that table. Will store employee’s id and table number.
* **“manage\_items” Table:** Table that shows the relationship between a manager and the items that that specific manager wants to manage. Will store the manager’s id and the item’s id.

**High-Level Database Schema:**

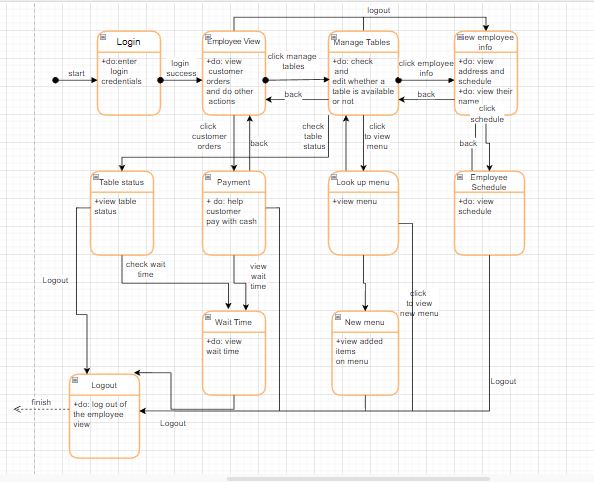


## Software Design

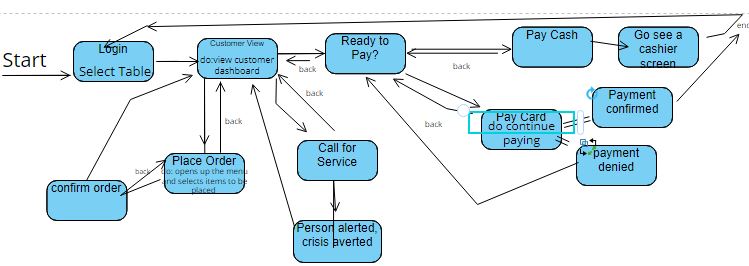
### State Machine Diagram: Manager (Kristin Cattell)



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

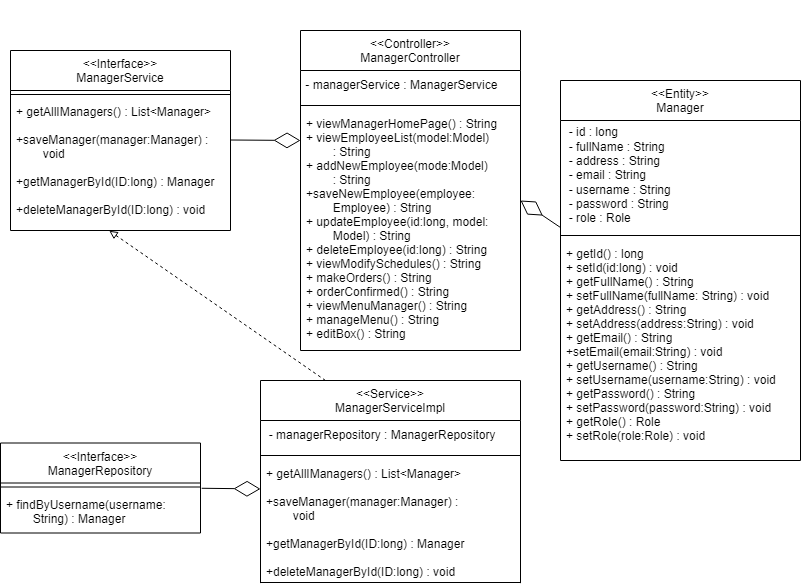


### State Machine Diagram: Customer (Ewan Allen)

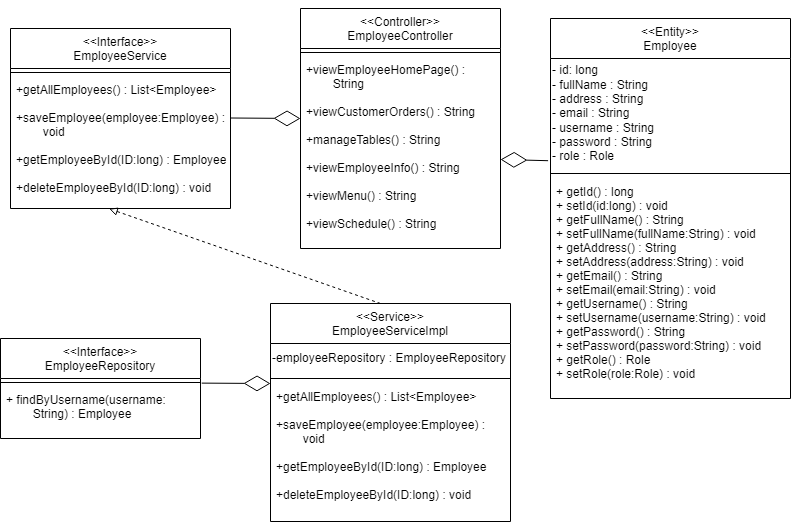


## UML Class Diagram

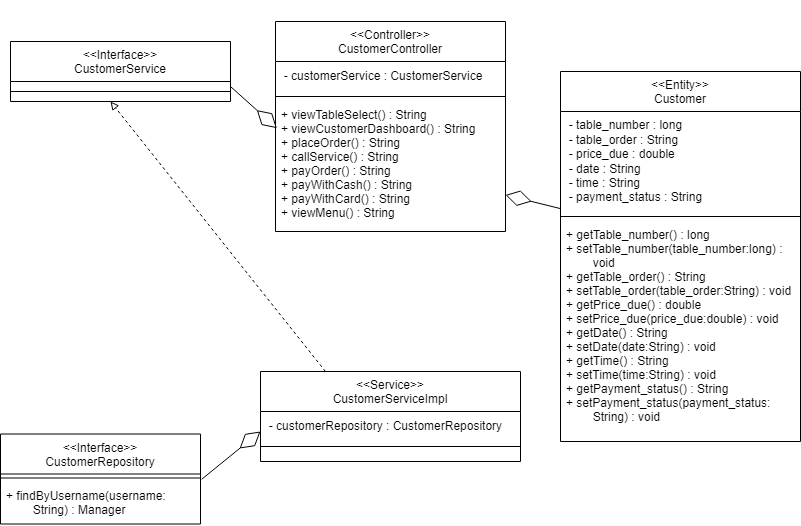
### UML Class Diagram: Manager



### UML Class Diagram: Employee

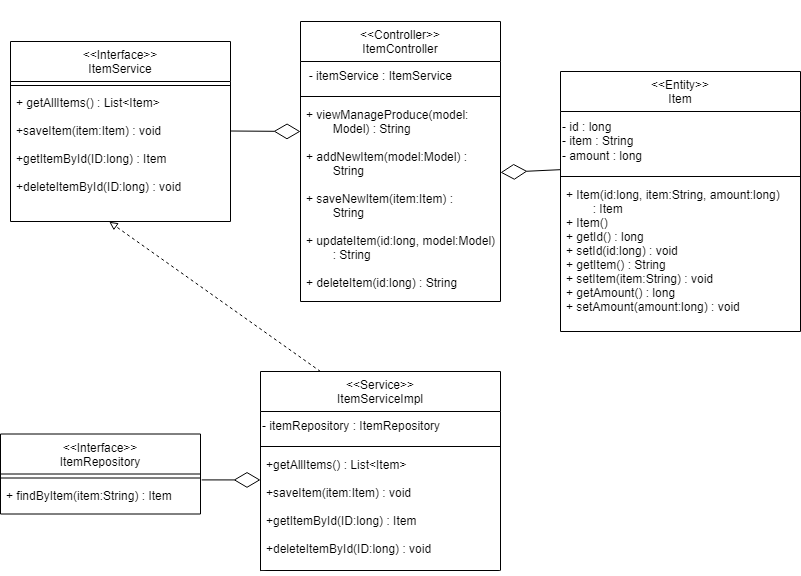


### UML Class Diagram: Customer

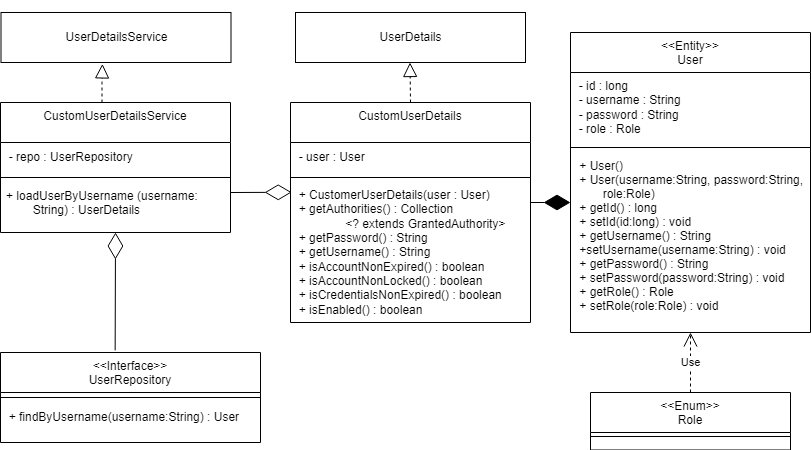


### UML Class Diagram: Others (Item, User, HomeController, SecurityConfigs)

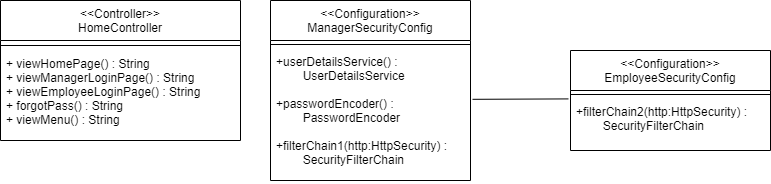
* **Item**:



* **User**:



* **HomeController & SecurityConfigs:**



Note: Some classes are currently empty due to not being able to implement certain functionalities yet. Once those functionalities are implemented within our system, the UMI Class Diagrams will be edited and filled in the future to match the system.