Design II - Spring 2022

CS 3704 - Intermediate Software Design

Vince Stevens, Tyler Yarow, Adam Oswald, Amanvir Singh, Aarya Shah

Overview

For our semester project we decided to design a restaurant automation software. Design II is the capstone design phase for this project; in this phase we combine the requirements and class diagrams we developed into a user interface to showcase how we see this software looking.

This document outlines our Design II submission. To accomplish our UI design, we utilized the third party software Balsamiq. The UI section is first, followed by the algorithm section, in which we detail some of the most challenging algorithms that we anticipate in our design and we follow them up with their associated pseudocode.

UI

Our user interface is only available in an app version. We will have 3 main users of our app: customers, employees and managers. Together, these three types of users make the automation process function, customers can view the menu, order, and reserve tables, while employees can view orders, schedule shifts, view payroll information, edit reservations, engage in training, and more. Finally, managers can do everything employees can do but with elevated privileges.

The next few pages detail our UI.

Algorithms

While developing this UI we identified several areas that could bottleneck our development progress. These areas were largely associated with algorithms that have a high development risk, algorithms that could be difficult to implement, and if developed incorrectly, could seriously increase the total development time and cost. In an effort to reduce this overall risk, we have outlined these algorithms and associated pseudo code below.

Login Verification - During application login, there are several steps to protect the login information of users and to ensure efficient verification. First, when a user attempts to login to our system, our system must communicate with a server on the backend that verifies if this username exists using an efficient search algorithm. If it does, the database sends back the public key to use to encrypt the password before sending it. Our system then encrypts and sends the password using an efficient algorithm. Finally, the login information sends the password to the database which uses an efficient encryption algorithm and the private key to un-encrypt the transferred password. Then it either confirms or denies the identity of the user.

```
Login(username, password) returns True/ False: (System-side method)
    Bool valid, public key = ValidateUser(username)
    If valid == true:
        EncryptPassword(password, public key)
        Return ValidatePassword(encrypted password)
    Else:
        Return false

ValidateUser(username) returns True/ False, public key: (Database-side method)
    BinarySearch database for username
    If username exists:
        Get associated public key
        Temporarily store private key, encrypted password
        Return true, public key
    Else:
        Return false, null

ValidatePassword(encrypted password) returns True/ False: (Database-side method)
    Grab stored private key, password
    Pass1 = Un-EncryptPassword(password_stored, private key)
    Pass2 = Un-EncryptPassword(password_stored, private key)
    If Pass1 == Pass2:
        Return true
    Else:
        Return false
```

Total Calculation - Our application has the capability for the customer to view their recent orders and receive an itemized receipt of a certain order. When a viewer requests a receipt for a certain order, the application will request the order from the backend and retrieve each listing that was ordered along with the general order information. The sum of the cost of the entire listing is the subtotal, which does not include any taxes or fees. The application will then retrieve the fees and taxes from the order, which is displayed separately. Finally, the total of the order gets shown, along with the general payment description that was used.

```
GenerateOrderView(OrderID)
    If Order = getOrder(OrderID) is successful:
        basicInfo = Order.getGeneralInfo()
        SalesLineList = Order.itemList()
        UIReceiptList = create new UIReceiptList object
        Iterate through SalesLineList:
            UIReceiptList.add(CurrSaleLineListing)

        SalesFooter = create new SalesFooter object
        Append "Subtotal: %d", Order.getSubtotal()
        Tariffs[] = Order.getTarrifs()
        For Tariff in Tarrifs:
            Append "Tax (%s): %d\n", Tariff.getDescription(),
    Order.getTariffAmount(Tariff) to SalesFooter
        Append result of Order.getPayment().getSummary() to SalesFooter
        Total = Order.getTotal()

        UIDrawOrder(basicInfo, UIReceiptList, SalesFooter, Total)
```

Employee Page (Scheduling, Orders Placed, Reservations) - This application is broken apart into multiple types of users. One of the main types of users are the employees. The employees have access to view their account, orders that have been placed, and modify reservations. Inside the view account, they can look at training videos, view their payroll, clock in and clock out for their working hours, and schedule when they want to come in and work. It is important to deconflict scheduling times between employees, so we have written pseudocode for the scheduling process, shown below the attached screenshots. Furthermore, we have included code to show the current reservations and orders, which are key pieces of the application.

```
Scheduling(String employeeName, int startTime, int endTime){
    If (employeeName == null)
        Return NULL
    For (int i = 0; i < timeTables.list; i++)
        Boolean timeFound = findTime(startTime, endTime)
        If (timeFound)
```

```
                Return true
            else
                System.error(“Time not found. Select new time”)
                Return false

OrdersPlaced(Orders[], int orderLength)
    If (orderLength == 0)
        return null
    else
        for (int i = 0; i < orderLength; i++)
            PrintList(Orders)

Reservations(Reservations[], int reservationsLength)
    If (reservationsLength == 0)
        return null
    else
        for (int i = 0; i < reservationsLength; i++)
            PrintList(Reservations)
```