Phase II: Class Diagram and Design Patterns

Nicole Garcia (nswanso6)

Design Patterns

Decorator Pattern:

The first design pattern I have identified to use for this project is the Decorator design pattern. The Decorator pattern is useful for modifying object functionality at runtime. It can add behaviors and roles to objects without changing their base structure. I will use this pattern for creating and modifying orders. Each order will have a base menu option, and within that, it can add various items to it that are available.

Observer Pattern:

The second design pattern I have identified to use for this project is the Observer design pattern. The observer pattern is useful for notifying the state of objects when changes have been made, It creates a one-to-many relationship between objects so that when changes are made to one, the rest will be notified. This pattern will be especially helpful when dealing with order status changes and notifying the customer of those updates. I will use the Observer pattern to handle the order management system.

Singleton Pattern:

The third design pattern I have identified to use for this project is the Singleton design pattern. The Singleton pattern can be used to restrict class initialization to only one class instance. This type of pattern is useful for handling authentication and ensuring that only one instance of a class is created for the authentication of user credentials and logins.

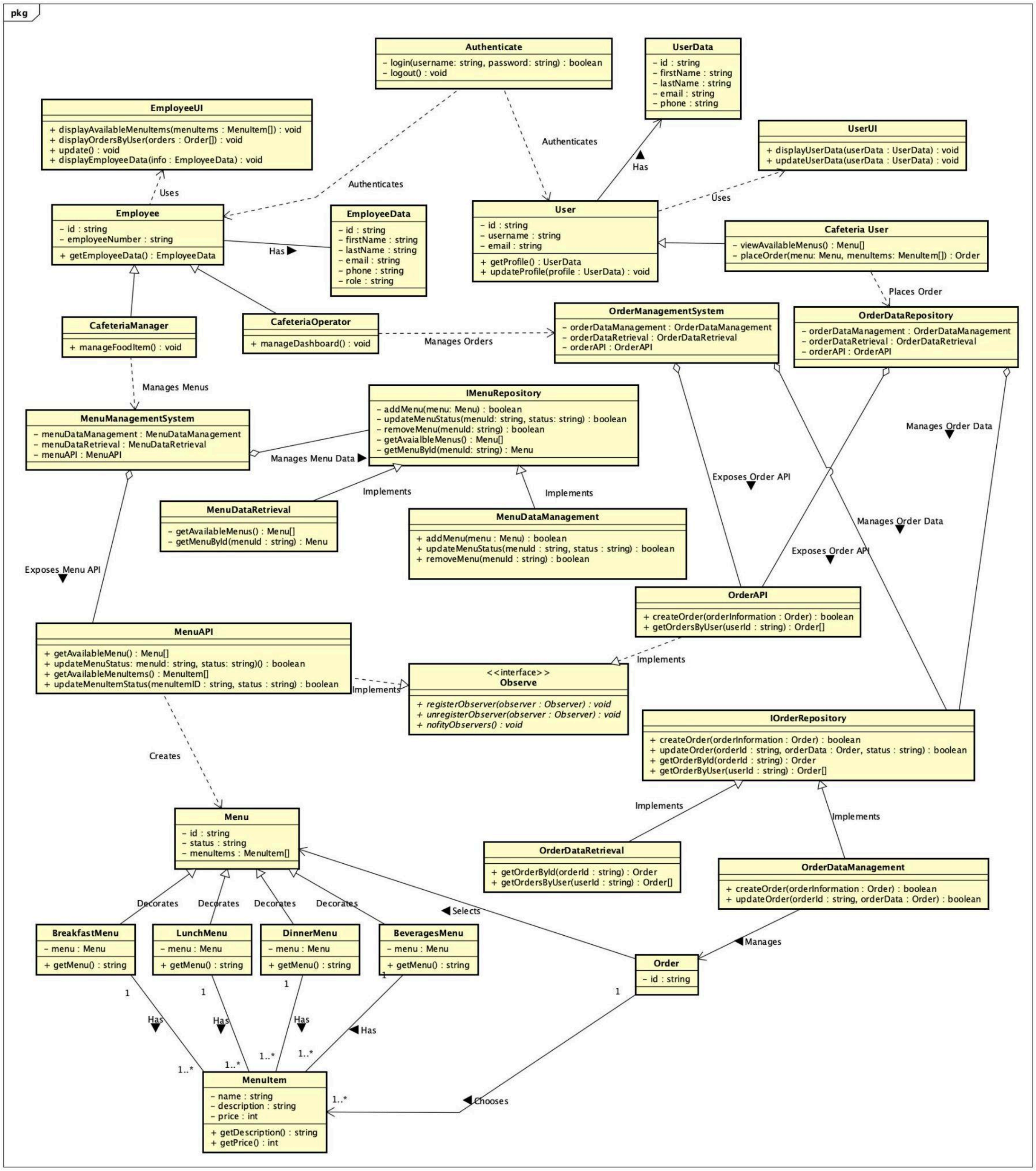
Factory Pattern:

The fourth design pattern I have identified to use for this project is the Factory design pattern. The Factory pattern provides the functionality of creating objects from a superclass which then modify the object types created. This pattern will be useful for creating menu classes with different menu items. The menu will be created using a certain type such as “breakfast” or “lunch”. The appropriate menu will be created and returned. This pattern could also be used for creating orders.

Iterator Pattern:

Another design pattern I have identified to use for this project is the Iterator design pattern. The Iterator pattern can be used to traverse through a group of objects. This pattern could be useful for traversing menu option items or customer orders.

Class Diagram


pkg
Authenticate
- login(username: string, password: string) : boolean
- logout() : void
UserData
- id : string
- firstName : string
- lastName : string
- email : string
- phone : string
EmployeeUI
+ displayAvailableMenuItems(menuItems : MenuItem[]) : void
+ displayOrdersByUser(orders : Order[]) : void
+ update() : void
+ displayEmployeeData(info : EmployeeData) : void
UserUI
+ displayUserData(userData : UserData) : void
+ updateUserData(userData : UserData) : void
Authenticates
Authenticates
Has
Uses
Uses
Employee
- id : string
- employeeNumber : string
+ getEmployeeData() : EmployeeData
Has
EmployeeData
- id : string
- firstName : string
- lastName : string
- email : string
- phone : string
- role : string
User
- id : string
- username : string
- email : string
+ getProfile() : UserData
+ updateProfile(profile : UserData) : void
Cafeteria User
- viewAvailableMenus() : Menu[]
- placeOrder(menu: Menu, menuItems: MenuItem[]) : Order
Places Order
CafeteriaManager
+ manageFoodItem() : void
CafeteriaOperator
+ manageDashboard() : void
Manages Orders
OrderManagementSystem
- orderDataManagement : OrderDataManagement
- orderDataRetrieval : OrderDataRetrieval
- orderAPI : OrderAPI
OrderDataRepository
- orderDataManagement : OrderDataManagement
- orderDataRetrieval : OrderDataRetrieval
- orderAPI : OrderAPI
Manages Menus
MenuManagementSystem
- menuDataManagement : MenuDataManagement
- menuDataRetrieval : MenuDataRetrieval
- menuAPI : MenuAPI
Manages Menu Data
IMenuRepository
- addMenu(menu: Menu) : boolean
- updateMenuStatus(menuId: string, status: string) : boolean
- removeMenu(menuId: string) : boolean
- getAvaialbleMenus() : Menu[]
- getMenuById(menuId: string) : Menu
Manages Order Data
Implements
Implements
Exposes Order API
MenuDataRetrieval
- getAvailableMenus() : Menu[]
- getMenuById(menuId : string) : Menu
MenuDataManagement
+ addMenu(menu : Menu) : boolean
+ updateMenuStatus(menuId : string, status : string) : boolean
+ removeMenu(menuId : string) : boolean
Manages Order Data
Exposes Order API
Exposes Menu API
OrderAPI
+ createOrder(orderInformation : Order) : boolean
+ getOrdersByUser(userId : string) : Order[]
MenuAPI
+ getAvailableMenu() : Menu[]
+ updateMenuStatus: menuId: string, status: string)() : boolean
+ getAvailableMenuItems() : MenuItem[]
+ updateMenuItemStatus(menuItemID : string, status : string) : boolean
Implements
Implements
<<interface>>
Observe
+ registerObserver(observer : Observer) : void
+ unregisterObserver(observer : Observer) : void
+ nofityObservers() : void
IOrderRepository
+ createOrder(orderInformation : Order) : boolean
+ updateOrder(orderId : string, orderData : Order, status : string) : boolean
+ getOrderById(orderId : string) : Order
+ getOrderByUser(userId : string) : Order[]
Creates
Implements
Implements
Menu
- id : string
- status : string
- menuItems : MenuItem[]
OrderDataRetrieval
+ getOrderById(orderId : string) : Order
+ getOrdersByUser(userId : string) : Order[]
OrderDataManagement
+ createOrder(orderInformation : Order) : boolean
+ updateOrder(orderId : string, orderData : Order) : boolean
Decorates
Decorates
Decorates
Decorates
Selects
BreakfastMenu
- menu : Menu
+ getMenu() : string
LunchMenu
- menu : Menu
+ getMenu() : string
DinnerMenu
- menu : Menu
+ getMenu() : string
BeveragesMenu
- menu : Menu
+ getMenu() : string
Manages
Order
- id : string
1
1
1
1
1
Has
Has
Has
Has
1..*
1..*
1..*
1..*
MenuItem
- name : string
- description : string
- price : int
+ getDescription() : string
+ getPrice() : int
1..*
Chooses

Classes created or modified:

1. Observer:

   The observer class was created to handle registering and unregistering observers and notifying registered observers of status updates. This will be used to communicate between the MenuAPI and OrderAPI classes.

2. Authentication:

The singleton pattern will be used to manage authentication and ensure that only one class is created to authenticate employees and users. The UserProfile class has been changed to UserData to be more consistent with the diagram for EmployeeData. An additional class UserUI was created for users to view and modify their data.

3. Factory and Decorator:

   The Factory pattern is used for creating menu objects in the MenuAPI class. The Decorator pattern will be used in the menu class to create different menu types for breakfast, lunch, dinner, and beverages. Each of these decorators will allow for the addition of specific menu items on the menu object.