**Design a Restaurant Management System**

**Let's design a Restaurant Management System**

**We'll cover the following:**

* [System Requirements](https://github.com/tssovi/grokking-the-object-oriented-design-interview/blob/master/object-oriented-design-case-studies/design-a-restaurant-management-system.md#system-requirements)
* [Use Case Diagram](https://github.com/tssovi/grokking-the-object-oriented-design-interview/blob/master/object-oriented-design-case-studies/design-a-restaurant-management-system.md#use-case-diagram)
* [Class Diagram](https://github.com/tssovi/grokking-the-object-oriented-design-interview/blob/master/object-oriented-design-case-studies/design-a-restaurant-management-system.md#class-diagram)
* [Activity Diagrams](https://github.com/tssovi/grokking-the-object-oriented-design-interview/blob/master/object-oriented-design-case-studies/design-a-restaurant-management-system.md#activity-diagrams)
* [Code](https://github.com/tssovi/grokking-the-object-oriented-design-interview/blob/master/object-oriented-design-case-studies/design-a-restaurant-management-system.md#code)

A Restaurant Management System is a software built to handle all restaurant activities in an easy and safe manner. This System will give the Restaurant management power and flexibility to manage the entire system from a single portal. The system allows the manager to keep track of available tables in the system as well as the reservation of tables and bill generation.

[](https://github.com/tssovi/grokking-the-object-oriented-design-interview/blob/master/media-files/restaurant-management-system.png)  
Restaurant Management System

**System Requirements**

We will focus on the following set of requirements while designing the Restaurant Management System:

1. The restaurant will have different branches.
2. Each restaurant branch will have a menu.
3. The menu will have different menu sections, containing different menu items.
4. The waiter should be able to create an order for a table and add meals for each seat.
5. Each meal can have multiple meal items. Each meal item corresponds to a menu item.
6. The system should be able to retrieve information about tables currently available to seat walk-in customers.
7. The system should support the reservation of tables.
8. The receptionist should be able to search for available tables by date/time and reserve a table.
9. The system should allow customers to cancel their reservation.
10. The system should be able to send notifications whenever the reservation time is approaching.
11. The customers should be able to pay their bills through credit card, check or cash.
12. Each restaurant branch can have multiple seating arrangements of tables.

**Use Case Diagram**

Here are the main Actors in our system:

**Receptionist:** Mainly responsible for adding and modifying tables and their layout, and creating and canceling table reservations. **Waiter:** To take/modify orders. **Manager:** Mainly responsible for adding new workers and modifying the menu. **Chef:** To view and work on an order. **Cashier:** To generate checks and process payments. **System:** Mainly responsible for sending notifications about table reservations, cancellations, etc.

Here are the top use cases of the Restaurant Management System:

* **Add/Modify tables:** To add, remove, or modify a table in the system.
* **Search tables:** To search for available tables for reservation.
* **Place order:** Add a new order in the system for a table.
* **Update order:** Modify an already placed order, which can include adding/modifying meals or meal items.
* **Create a reservation:** To create a table reservation for a certain date/time for an available table.
* **Cancel reservation:** To cancel an existing reservation.
* **Check-in:** To let the guest check in for their reservation.
* **Make payment:** Pay the check for the food.

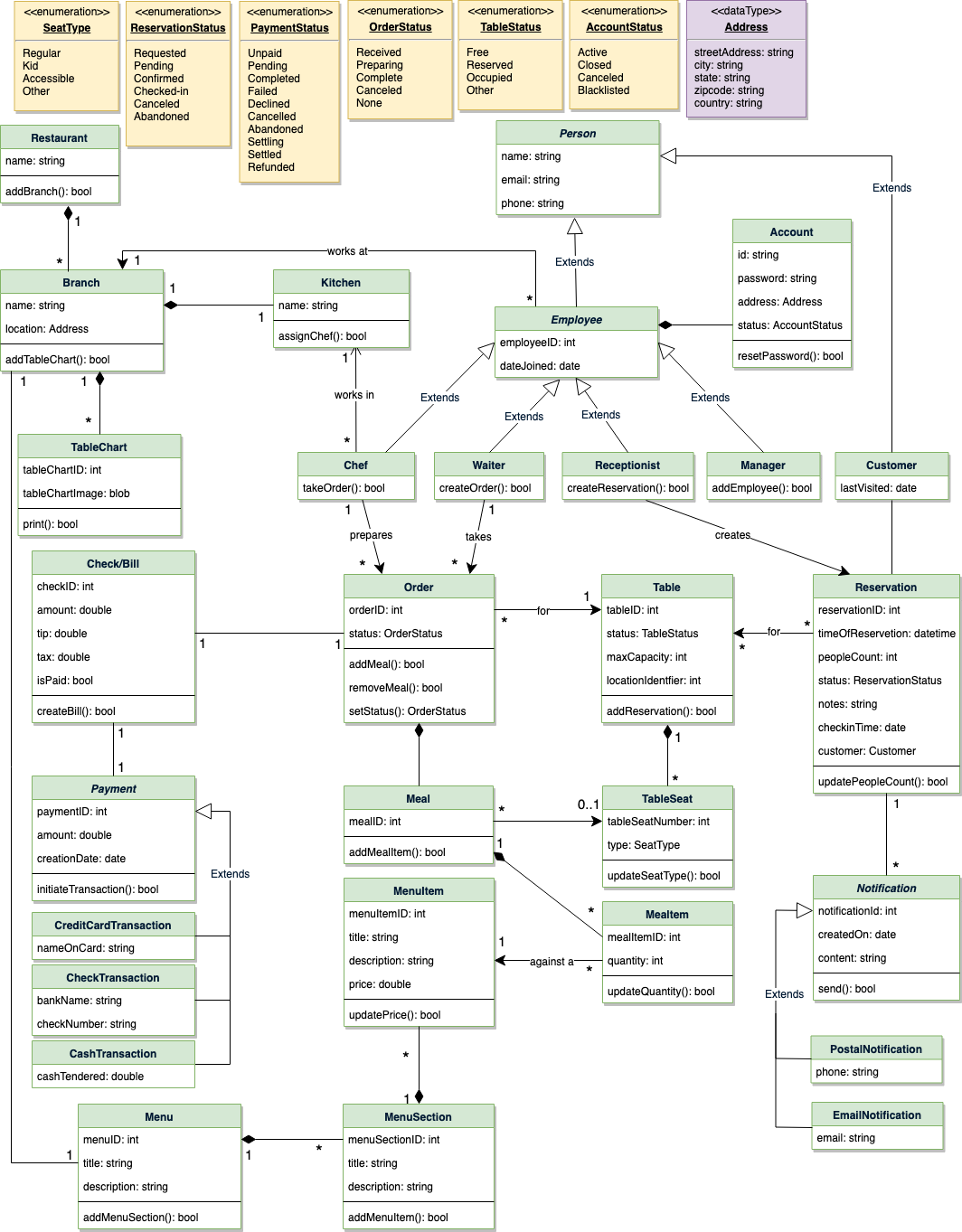
Here is the use case diagram of our Restaurant Management System:

Use Case Diagram for Restaurant Management System

**Class Diagram**

Here is the description of the different classes of our Restaurant Management System:

* **Restaurant:** This class represents a restaurant. Each restaurant has registered employees. The employees are part of the restaurant because if the restaurant becomes inactive, all its employees will automatically be deactivated.
* **Branch:** Any restaurants can have multiple branches. Each branch will have its own set of employees and menus.
* **Menu:** All branches will have their own menu.
* **MenuSection and MenuItem:** A menu has zero or more menu sections. Each menu section consists of zero or more menu items.
* **Table and TableSeat:** The basic building block of the system. Every table will have a unique identifier, maximum sitting capacity, etc. Each table will have multiple seats.
* **Order:** This class encapsulates the order placed by a customer.
* **Meal:** Each order will consist of separate meals for each table seat.
* **Meal Item:** Each Meal will consist of one or more meal items corresponding to a menu item.
* **Account:** We’ll have different types of accounts in the system, one will be a receptionist to search and reserve tables and the other, the waiter will place orders in the system.
* **Notification:** Will take care of sending notifications to customers.
* **Bill:** Contains different bill-items for every meal item.

[](https://github.com/tssovi/grokking-the-object-oriented-design-interview/blob/master/media-files/rms-class-diagram.png)  
Class Diagram for Restaurant Management System

UML for Restaurant Management System

**Activity Diagrams**

**Place order:** Any waiter can perform this activity. Here are the steps to place an order:

Activity Diagram for Restaurant Management System Place Order

**Make a reservation:** Any receptionist can perform this activity. Here are the steps to make a reservation:

Activity Diagram for Restaurant Management System Make Reservation

**Cancel a reservation:** Any receptionist can perform this activity. Here are the steps to cancel a reservation:

Activity Diagram for Restaurant Management System Cancel Reservation

**Code**

Here is the high-level definition for the classes described above.

**Enums, data types, and constants:** Here are the required enums, data types, and constants:

from enum import Enum

class ReservationStatus(Enum):

REQUESTED, PENDING, CONFIRMED, CHECKED\_IN, CANCELED, ABANDONED = 1, 2, 3, 4, 5, 6

class SeatType(Enum):

REGULAR, KID, ACCESSIBLE, OTHER = 1, 2, 3, 4

class OrderStatus(Enum):

RECEIVED, PREPARING, COMPLETED, CANCELED, NONE = 1, 2, 3, 4, 5

class TableStatus(Enum):

FREE, RESERVED, OCCUPIED, OTHER = 1, 2, 3, 4

class AccountStatus(Enum):

ACTIVE, CLOSED, CANCELED, BLACKLISTED, BLOCKED = 1, 2, 3, 4, 5

class PaymentStatus(Enum):

UNPAID, PENDING, COMPLETED, FILLED, DECLINED, CANCELLED, ABANDONED, SETTLING, SETTLED, REFUNDED = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

class Address:

def \_\_init\_\_(self, street, city, state, zip\_code, country):

self.\_\_street\_address = street

self.\_\_city = city

self.\_\_state = state

self.\_\_zip\_code = zip\_code

self.\_\_country = country

**Account, Person, Employee, Receptionist, Manager, and Chef:** These classes represent the different people that interact with our system:

from abc import ABC

from datetime import datetime

from .constants import \*

# For simplicity, we are not defining getter and setter functions. The reader can

# assume that all class attributes are private and accessed through their respective

# public getter methods and modified only through their public methods function.

class Account:

def \_\_init\_\_(self, id, password, address, status=AccountStatus.Active):

self.\_\_id = id

self.\_\_password = password

self.\_\_address = address

self.\_\_status = status

def reset\_password(self):

None

class Person(ABC):

def \_\_init\_\_(self, name, email, phone):

self.\_\_name = name

self.\_\_email = email

self.\_\_phone = phone

class Employee(ABC, Person):

def \_\_init\_\_(self, id, account, name, email, phone):

super().\_\_init\_\_(name, email, phone)

self.\_\_employee\_id = id

self.\_\_date\_joined = datetime.date.today()

self.\_\_account = account

class Receptionist(Employee):

def \_\_init\_\_(self, id, account, name, email, phone):

super().\_\_init\_\_(id, account, name, email, phone)

def create\_reservation(self):

None

def search\_customer(self, name):

None

class Manager(Employee):

def \_\_init\_\_(self, id, account, name, email, phone):

super().\_\_init\_\_(id, account, name, email, phone)

def add\_employee(self):

None

class Chef(Employee):

def \_\_init\_\_(self, id, account, name, email, phone):

super().\_\_init\_\_(id, account, name, email, phone)

def take\_order(self):

None

**Restaurant, Branch, Kitchen, TableChart:** These classes represent the top-level classes of the system

class Kitchen:

def \_\_init\_\_(self, name):

self.\_\_name = name

self.\_\_chefs = []

def assign\_chef(self, chef):

None

class Branch:

def \_\_init\_\_(self, name, location, kitchen):

self.\_\_name = name

self.\_\_location = location

self.\_\_kitchen = kitchen

def add\_table\_chart(self):

None

class Restaurant:

def \_\_init\_\_(self, name):

self.\_\_name = name

self.\_\_branches = []

def add\_branch(self, branch):

None

class TableChart:

def \_\_init\_\_(self, id):

self.\_\_table\_chart\_id = id

self.\_\_table\_chart\_image = []

def print(self):

None

**Table, TableSeat, and Reservation:** Each table can have multiple seats and customers can make reservations for tables:

from datetime import datetime

from .constants import \*

class Table:

def \_\_init\_\_(self, id, max\_capacity, location\_identifier, status=TableStatus.FREE):

self.\_\_table\_id = id

self.\_\_max\_capacity = max\_capacity

self.\_\_location\_identifier = location\_identifier

self.\_\_status = status

self.\_\_seats = []

def is\_table\_free(self):

None

def add\_reservation(self):

None

def search(self, capacity, start\_time):

# return all tables with the given capacity and availability

None

class TableSeat:

def \_\_init\_\_(self):

self.\_\_table\_seat\_number = 0

self.\_\_type = SeatType.REGULAR

def update\_seat\_type(self, seat\_type):

None

class Reservation:

def \_\_init\_\_(self, id, people\_count, notes, customer):

self.\_\_reservation\_id = id

self.\_\_time\_of\_reservation = datetime.now()

self.\_\_people\_count = people\_count

self.\_\_status = ReservationStatus.REQUESTED

self.\_\_notes = notes

self.\_\_checkin\_time = datetime.now()

self.\_\_customer = customer

self.\_\_tables = []

self.\_\_notifications = []

def update\_people\_count(self, count):

None

**Menu, MenuSection, and MenuItem:** Each restaurant branch will have its own menu, each menu will have multiple menu sections, which will contain menu items:

class MenuItem:

def \_\_init\_\_(self, id, title, description, price):

self.\_\_menu\_item\_id = id

self.\_\_title = title

self.\_\_description = description

self.\_\_price = price

def update\_price(self, price):

None

class MenuSection:

def \_\_init\_\_(self, id, title, description):

self.\_\_menu\_section\_id = id

self.\_\_title = title

self.\_\_description = description

self.\_\_menu\_items = []

def add\_menu\_item(self, menu\_item):

None

class Menu:

def \_\_init\_\_(self, id, title, description):

self.\_\_menu\_id = id

self.\_\_title = title

self.\_\_description = description

self.\_\_menu\_sections = []

def add\_menu\_section(self, menu\_section):

None

def print(self):

None

**Order, Meal, and MealItem:** Each order will have meals for table seats:

from datetime import datetime

class MealItem:

def \_\_init\_\_(self, id, quantity, menu\_item):

self.\_\_meal\_item\_id = id

self.\_\_quantity = quantity

self.\_\_menu\_item = menu\_item

def update\_quantity(self, quantity):

None

class Meal:

def \_\_init\_\_(self, id, seat):

self.\_\_meal\_id = id

self.\_\_seat = seat

self.\_\_menu\_items = []

def add\_meal\_item(self, meal\_item):

None

class Check():

def \_\_init\_\_(self):

None

class Order:

def \_\_init\_\_(self, id, status, table, waiter, chef):

self.\_\_order\_id = id

self.\_\_OrderStatus = status

self.\_\_creation\_time = datetime.now()

self.\_\_meals = []

self.\_\_table = table

self.\_\_waiter = waiter

self.\_\_chef = chef

self.\_\_check = Check()

def add\_meal(self, meal):

None

def remove\_meal(self, meal):

None

def get\_status(self):

return self.\_\_OrderStatus

def set\_status(self, status):

None