Restaurant Reservation System Design Document

PROBLEM #1 Your job is to create a reservation system for a restaurant. The restaurant has N tables. You need to be able to provide following functionality for this system. Let user reserve a table or clear a reservation.

- In a restaurant, there are multiple tables with varying seating capacities.
- A table of seating capacity of n can be booked for m people of a group such that m <= n.
- A table can be re-booked for the same day only if there is a minimum T duration difference between 2 bookings.
- Booking can be done in fixed hours when the restaurant operates.

Task: First, start with a design and identify the classes, data members and interfaces needed and how they will interact with each other. Define all the entities need for this application and the assumptions you are going to make. And Implement the classes and interfaces defined above.

PROBLEM # 2 (Extending your solution to add additional features via Feature Flags) Suggest other "optional" features you would like to add to your reservation system. Depending on the client vendor, you should be able to enable/disable these features (Think of these features which would enhance your solution for customers)

PROPOSED SYSTEM

1.1 Required Functions:

- Table Capacity: The system will show multiple tables with varying seating capacities.
- Table Booking: Customers of a group of m people could only book table of seating capacity of n that n>=m.
- Time Interval: The system will allow re-booking of a table for the same day only if there is a minimum T duration difference between 2 bookings.
- Time Display: The system will allow customers to book table in fixed hours when the restaurant operates.

1.2 Accompanying Functions:

- Customer Information: The system will require customers to enter name and phone number for contact.
- Time Limitation: The system will not allow customers to enter a time in the past.
- Reservation Status: The system will display if the reservation is successful once the customers submitted.

1.3 Featured Functions:

- Reservation Summarizing: The manager could check the overall reservation status of a specific date from admin view.
- Reservation Display: The manager could check all the customers' reservation information including name, phone number, guest number, reserved table and reserved time.
- Table Adjustment: The manager could add tables of varied seating capacities for customers' reservation needs based on situation.
- Utilization Status: The manager could check utilization status of all the tables on any date.

SYSTEM MODELS

2.1 User Cases:

Name:	Reserve a table
Actor:	Customer, server
Entry Conditions:	User clicks "Reservation a Table"
Flow of Events:	1. Customer enter name, phone number, number of guests and reservation time. 2. Customer click "create reservation" 3. System locks the database 4. System modifies database to reflect reservation 5. System unlocks the database
Exit Conditions:	• Reservation failed: wrong time/ table is taken at the time • Reservation succeeded

Name:	Check summarized reservation of one day
Actor:	Manager, server
Entry Conditions:	User clicks "Check Reservations"
Flow of Events:	1. Manager enters specific date to check 2. Server Queried for reservations on that date 3. Front end displays reservation data on that date
Exit Conditions:	• Manager clicks "Back to Admin"

Name:	Add tables to the restaurant reservation system
Actor:	Manager, server
Entry Conditions:	User clicks “Check Tables”
Flow of Events:	1. Manager enters capacity of table to add 2. Manager clicks “Add Table” 3. System locks the database 4. System modifies database to reflect table status 5. System unlocks the database
Exit Conditions:	• Manager clicks “Back to Admin”

OBJECT MODELS

3.1 Class Identification:

This part of code are included in models.py in the app

```
class Guest(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(64), index=True)
    phone_number = db.Column(db.String(64), index=True, unique=True)
```

```
class Table(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    capacity = db.Column(db.Integer, index=True)
```

```
class Reservation(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    guest_id = db.Column(db.Integer, db.ForeignKey('guest.id'))
    guest = db.relationship('Guest')
    table_id = db.Column(db.Integer, db.ForeignKey('table.id'))
    table = db.relationship('Table')
    num_guests = db.Column(db.Integer, index=True)
    reservation_time = db.Column(db.DateTime, index=True)
```

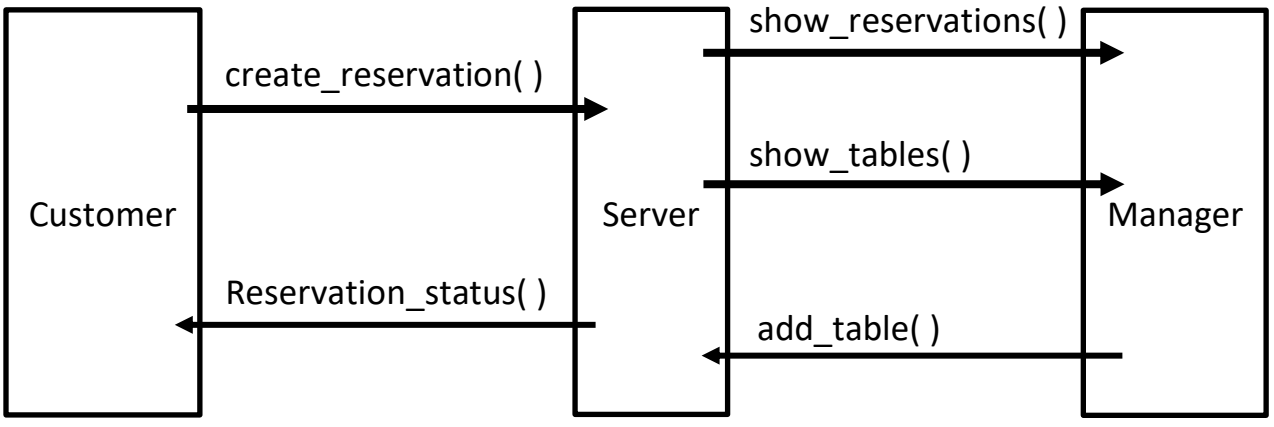
This part of code are included in forms.py in the app

```
class ReservationForm(Form):
    guest_name = StringField('guest_name', validators=[DataRequired()])
    guest_phone = StringField('guest_phone', validators=[DataRequired()])
    num_guests = SelectField('num_guests', coerce=int, choices = [(x, x) for x in range(1, MAX_TABLE_CAPACITY)])
    reservation_datetime = DateTimeField('reservation_datetime', default=datetime.now(),validators=[DataRequired()])
```

```
class ShowReservationsOnDateForm(Form):
    reservation_date = DateField('reservation_date', default=datetime.now())
```

```
class AddTableForm(Form):
    table_capacity = SelectField('table_capacity', coerce=int, choices = [(x, x) for x in range(1, MAX_TABLE_CAPACITY)])
```

3.2 Interaction Diagram:



3.3 Implementation:

This part of code are included in controller.py in the app

```
def create_reservation(form_data):
```

This part of code are included in views.py in the app

```
def reservation_status():
```

```
def show_tables():
```

```
def show_reservations():
```

SYSTEM APPLICATION

4.1 Running Environment:

In the coding and testing of the system, I use flask as framework to build all the design and implementations described above.

```
$ virtualenv flask

$ cd flask

$ source bin/activate

$ pip install flask

$ pip install Flask-SQLAlchemy

$ pip install flask-wtf

$ python run.py
```

4.2 Application Testing: