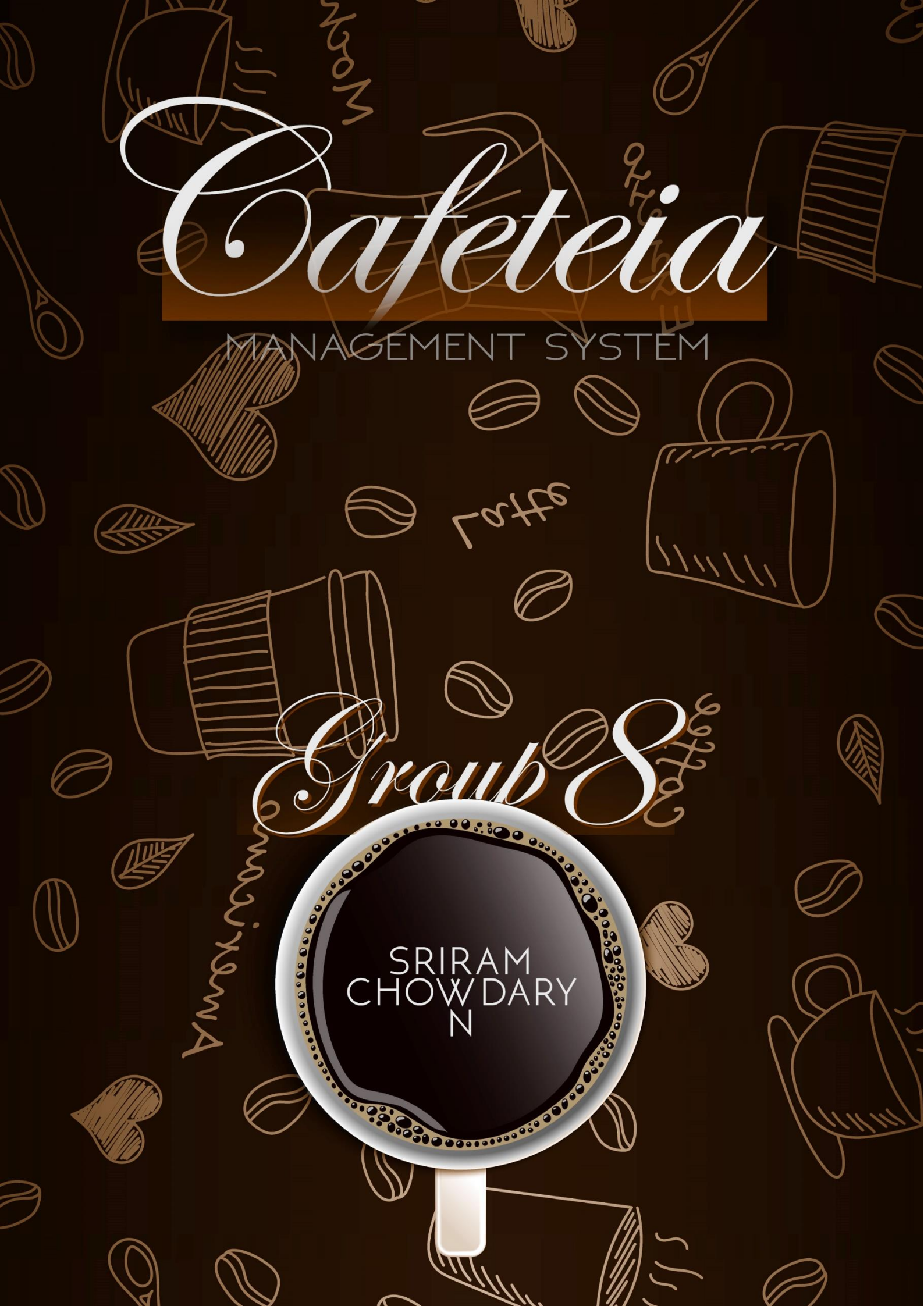
Cafeteia

MANAGEMENT SYSTEM



Group 8

SRIRAM CHOWDARY N

Chettinad Vidyashram

(Affiliated to Central Board of Secondary Education, New Delhi)

(Chettinad House, R.A.Puram, Chennai – 600 028)

COMPUTER SCIENCE

Certified to be the Bonafide Record of work done by

_____ of Std XII Sec ____

in the Computer Science Lab of the CHETTINAD VIDYASHRAM, CHENNAI, during the year 2021 – 2022.

Date: Teacher-in-charge

REGISTER NO. _____

Submitted for All India Senior Secondary Practical Examination in

Computer Science held on _____at

Chettinad Vidyashram, Chennai – 600 028.

Principal *Internal Examiner* *External Examiner*

ACKNOWLEDGEMENT

I would like to express my sincere thanks to

Meena Aunty, Principal Mrs. S.Amudha lakshmi

for their encouragement and support to work on

this Project. I am grateful to my computer

science

teacher UMA MAGESWARI R and to the

computer science department for the constant

guidance and support to complete the project

TABLE OF CONTENT

OVERVIEW OF PYTHON

Python is a **computer programming language often used to build websites and software, automate tasks, and conduct data analysis**. Python is a general-purpose language, meaning it can be used to create a variety of different programs and isn't specialized for any specific problems.

It is a high-level, **interpreted, interactive and object-oriented scripting** language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

1. **Python is Interpreted** − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
2. **Python is Interactive** − You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
3. **Python is Object-Oriented** − Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
4. **Python is a Beginner's Language** − Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

Python's features include −

- **Easy-to-learn** − Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
- **Easy-to-read** − Python code is more clearly defined and visible to the eyes.
- **Easy-to-maintain** − Python's source code is fairly easy-to-maintain.
- **A broad standard library** − Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- **Interactive Mode** − Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- **Portable** − Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- **Databases** − Python provides interfaces to all major commercial databases.

- **GUI Programming** − Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
- **Scalable** − Python provides a better structure and support for large programs than shell scripting.

PROJECT DESCRIPTION

Our project titled “CVTERIA” provides a GUI based system to efficiently manage a cafeteria. We have used a combination of Python and MySQL for the back-end code and have designed a simple yet robust front-end GUI using Tkinter.

This application allows both the staff at a cafeteria and the cafeteria manager to manage records of the customers digitally. It includes features like adding customers, updating customer details, deleting customer details, and ability to view the list of customers including different sorting capabilities. For security purposes we have built a simple yet powerful password validation function. Our appropriate pop-up messages for various scenarios helps the user understand what he or she has done to the database.

The application has multiple validations at user enterable fields and data base level exceptions. We have mapped all the errors with user friendly messages.

The system runs without any glitches at an optimal speed. Being an open-source application, this is a cost-effective solution for small cafeterias.

FUNCTIONS USED

Functions Used	Function Description
connect()	Used to connect mysql to python
setup()	To create tables and declare variables
seed()	To create data inside each table, like menu etc.
customer_create()	To create customer
customer_update()	To update customer
customer_delete()	To delete customer
bill_delete()	To delete bill
order_delete()	To delete an order
customer_get_all()	To fetch all customers from database
customer_get_by_id()	To get customer information for a given customer ID

order_header_create()	To create order header
order_header_update()	To update order header
order_detail_update()	To update order details
order_detail_create()	To create order details
order_header_get_all()	To fetch all order headers from the database
order_header_get_upaid()	To get order header on paying the bill
order_header_get_by_id()	To get order header by order ID
order_detail_get_all()	To fetch all order details from the database
order_detail_get_by_id()	To get order details by order ID.

order_detail_delete_by_id()	To delete order details based on given order ID.
item_get_all()	To Get all menu items from the database
item_get_by_category_id()	To get all menu items from the database based on category ID.

table_get_all()	To get all tables from the database
table_reservation_get_all()	To get all table reservations from the database
table_reservation_get_by_id()	To get table reservations from the database based on ID
table_reservation_create()	To create a new table reservation
table_reservation_update()	To update table reservation details
table_reservation_delete()	To delete table reservations
payment_mode_get_all()	To get all payment modes from the database.

bill_create()	To create a new bill.
bill_update()	To update bill details.

Bill_get_all()	To get all bills from the database
bill_get_by_id()	To get all bills from the database based on ID.
bill_get_buy_order_header_id()	To get all bills from database based on order header ID
bill_get_buy_customer_id()	To get all bills from database based on customer ID
category_get_all()	To get all categories from the database

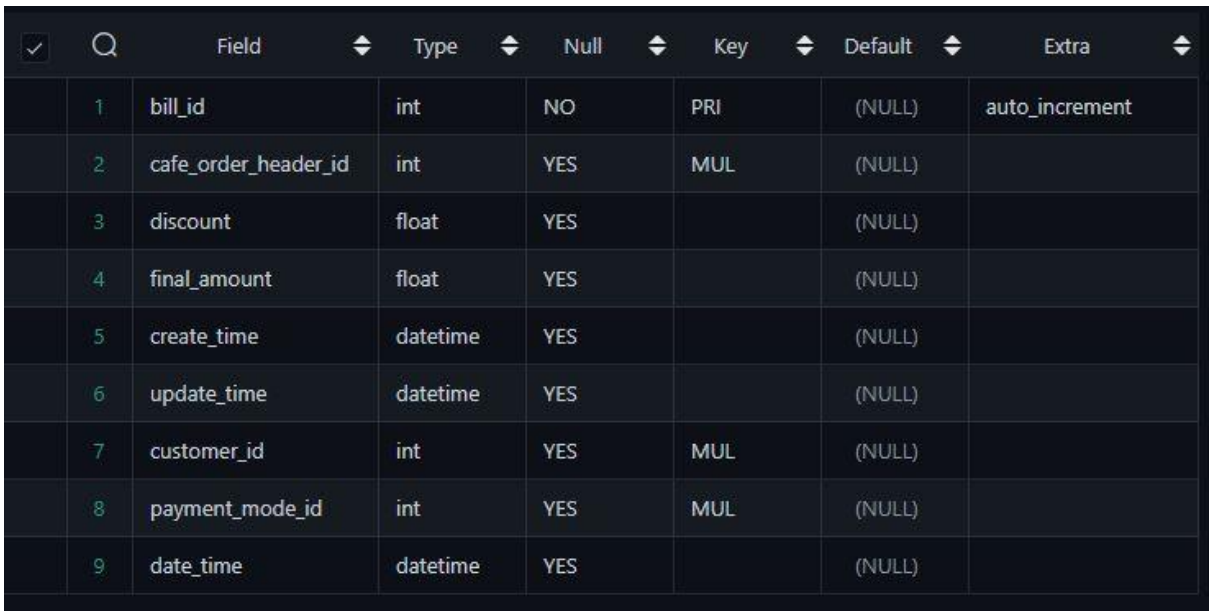
TABLES USED

BILL

Description:

The bill table is used to store the details of every bill. It has attributes like bill_id, discount,final_amount,payment_mode_id,date_time etc. This table can be accessed only by the owner of the café in order to manage the cafeteria.

Structure:



	Field	Type	Null	Key	Default	Extra
1	bill_id	int	NO	PRI	(NULL)	auto_increment
2	cafe_order_header_id	int	YES	MUL	(NULL)	
3	discount	float	YES		(NULL)	
4	final_amount	float	YES		(NULL)	
5	create_time	datetime	YES		(NULL)	
6	update_time	datetime	YES		(NULL)	
7	customer_id	int	YES	MUL	(NULL)	
8	payment_mode_id	int	YES	MUL	(NULL)	
9	date_time	datetime	YES		(NULL)	

Café_order_detail

Description:

The table contains the details of every order that has been placed at our café. It has attributes like cafe_order_detail_id, menu_id, qty, unit_price etc.

Structure:

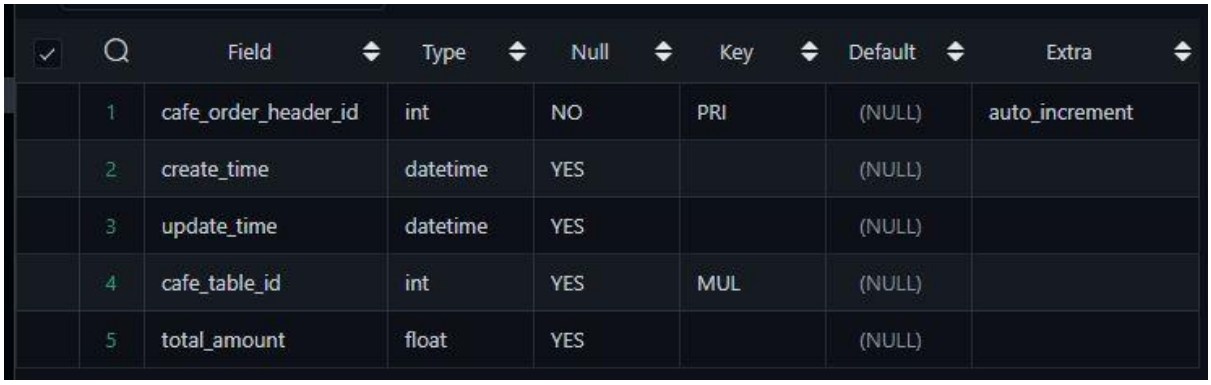
	Field	Type	Null	Key	Default	Extra
1	cafe_order_detail_id	int	NO	PRI	(NULL)	auto_increment
2	cafe_order_header_id	int	YES	MUL	(NULL)	
3	create_time	datetime	YES		(NULL)	
4	update_time	datetime	YES		(NULL)	
5	menu_id	int	YES	MUL	(NULL)	
6	qty	int	YES		(NULL)	
7	unit_price	float	YES		(NULL)	
8	gst	float	YES		(NULL)	
9	amount	float	YES		(NULL)	

Cafe_order_header

Description:

This table stores details of order headers and total amount of each order. It has attributes like café_table_id, total_amount etc.

Structure:



	Field	Type	Null	Key	Default	Extra
1	cafe_order_header_id	int	NO	PRI	(NULL)	auto_increment
2	create_time	datetime	YES		(NULL)	
3	update_time	datetime	YES		(NULL)	
4	cafe_table_id	int	YES	MUL	(NULL)	
5	total_amount	float	YES		(NULL)	

Cafe_table

Description:

This table is used to store the details of the tables at our café. It has attributes like café_table_id,café_table_number etc.

Structure:

	Field	Type	Null	Key	Default	Extra
1	cafe_table_id	int	NO	PRI	(NULL)	auto_increment
2	create_time	datetime	YES		(NULL)	
3	update_time	datetime	YES		(NULL)	
4	no_of_seats	int	YES		(NULL)	
5	cafe_table_number	varchar(255)	YES		(NULL)	
6	location	varchar(255)	YES		(NULL)	

Category

Description:

This table stores the categories to which each menu item belongs to. It has attributes like category_id, category_name etc.

Structure:

	Field	Type	Null	Key	Default	Extra
1	category_id	int	NO	PRI	(NULL)	auto_increment
2	create_time	datetime	YES		(NULL)	
3	update_time	datetime	YES		(NULL)	
4	category_name	varchar(255)	YES		(NULL)	

Customer

Description:

This table is used to store the details of the customers who have created accounts at our café.It has attrbutes like customer_id,customer_name etc.

Structure:

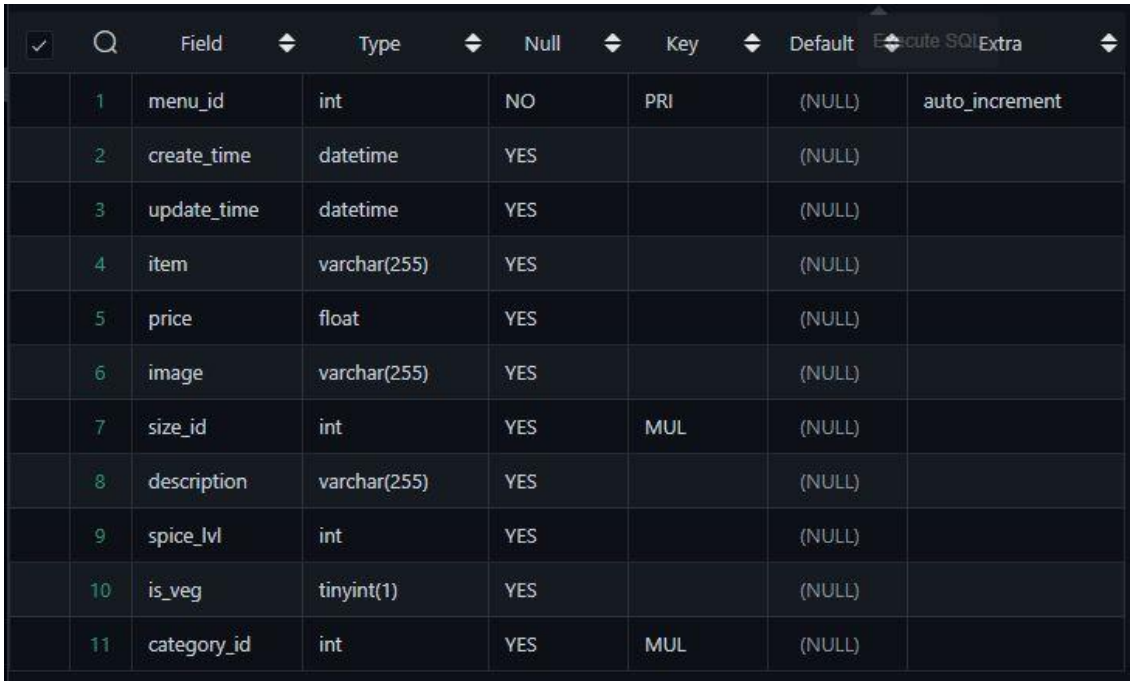
	Field	Type	Null	Key	Default	Extra
1	customer_id	int	NO	PRI	(NULL)	auto_increment
2	create_time	datetime	YES		(NULL)	
3	update_time	datetime	YES		(NULL)	
4	customer_name	varchar(255)	YES		(NULL)	
5	customer_phone_numbe	varchar(255)	YES		(NULL)	
6	gender	tinyint(1)	YES		(NULL)	
7	email_id	varchar(255)	YES		(NULL)	
8	DOB	date	YES		(NULL)	
9	DOA	date	YES		(NULL)	

Menu

Description:

This table stores all our menu items. It has attributes like menu_id, price,item etc.

Structure:



	Field	Type	Null	Key	Default	Extra
1	menu_id	int	NO	PRI	(NULL)	auto_increment
2	create_time	datetime	YES		(NULL)	
3	update_time	datetime	YES		(NULL)	
4	item	varchar(255)	YES		(NULL)	
5	price	float	YES		(NULL)	
6	image	varchar(255)	YES		(NULL)	
7	size_id	int	YES	MUL	(NULL)	
8	description	varchar(255)	YES		(NULL)	
9	spice_lvl	int	YES		(NULL)	
10	is_veg	tinyint(1)	YES		(NULL)	
11	category_id	int	YES	MUL	(NULL)	

Payment mode

Description:

This table stores the different payment modes our café offers. It has attributes like payment_mode_id, payment_mode_name etc.

Structure:

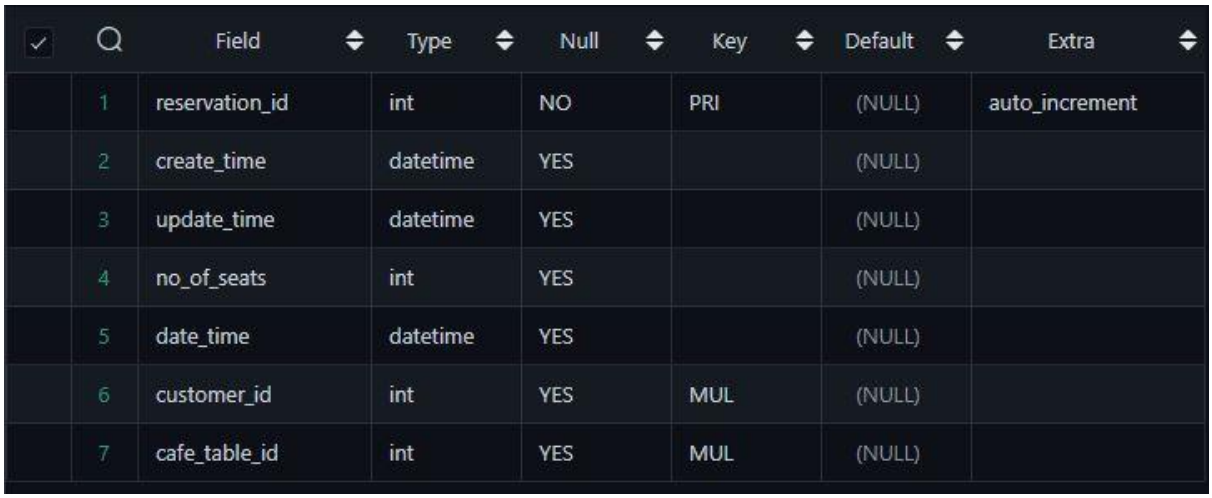
	Field	Type	Null	Key	Default	Extra
1	payment_mode_id	int	NO	PRI	(NULL)	auto_increment
2	create_time	datetime	YES		(NULL)	
3	update_time	datetime	YES		(NULL)	
4	payment_mode_name	varchar(255)	YES		(NULL)	

Reservation

Description:

This table stores all the reservations made by customers at our café. It has attributes like reservation_id, no_of_seats etc.

Structure:

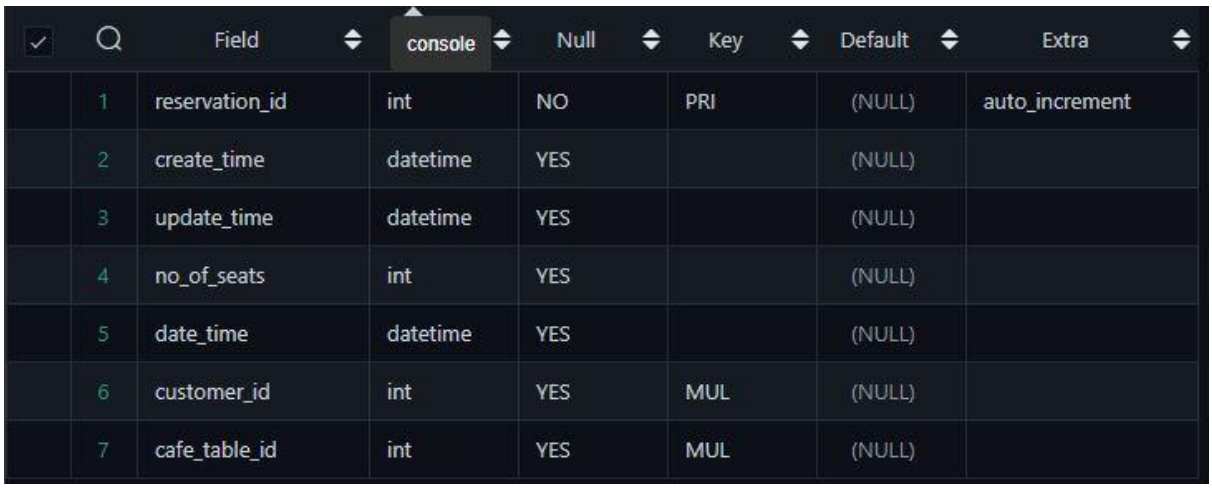


	Field	Type	Null	Key	Default	Extra
1	reservation_id	int	NO	PRI	(NULL)	auto_increment
2	create_time	datetime	YES		(NULL)	
3	update_time	datetime	YES		(NULL)	
4	no_of_seats	int	YES		(NULL)	
5	date_time	datetime	YES		(NULL)	
6	customer_id	int	YES	MUL	(NULL)	
7	cafe_table_id	int	YES	MUL	(NULL)	

Size

Description:This tables stores the different sizes that our café offers for each item. It has attributes like size_id, size_name..

Structure:



	Field	console	Null	Key	Default	Extra
1	reservation_id	int	NO	PRI	(NULL)	auto_increment
2	create_time	datetime	YES		(NULL)	
3	update_time	datetime	YES		(NULL)	
4	no_of_seats	int	YES		(NULL)	
5	date_time	datetime	YES		(NULL)	
6	customer_id	int	YES	MUL	(NULL)	
7	cafe_table_id	int	YES	MUL	(NULL)	

db.py

```
import mysql.connector as bot
from model.category import Category

from model.customer import Customer
from model.order_header import OrderHeader
from model.order_detail import OrderDetail
from model.item import Item
from model.table import Table
from model.table_reservation import TableReservation
from model.payment_mode import PaymentMode
from model.bill import Bill

mydb = None

class Database():
    def connect(self):
        mydb = bot.connect(
            host = 'localhost',
            user = 'root',
            password = 'Tiger@123',
            database = 'cvteria'
        )

        if mydb:
            print('connected to the db successfully!!')
        else:
            print('could not connect to DB')
        return mydb

    def setup(self):
```

```
    #CREATE DATBASE
    con =
bot.connect(host="localhost",user="root",password="Tiger@123",charset='utf8
')
    cur = con.cursor()
    cur.execute("""
      CREATE DATABASE IF NOT EXISTS cvteria
      """)

    mydb = self.connect()
    mycursor = mydb.cursor()

    #CREATE TABLES
    #size
    mycursor.execute("""
      CREATE TABLE IF NOT EXISTS
      size(
        size_id int NOT NULL PRIMARY KEY AUTO_INCREMENT COMMENT
'Primary Key',
        create_time DATETIME COMMENT 'Create Time',
        update_time DATETIME COMMENT 'Update Time',
        size_name VARCHAR(255)
      );
      """)

    #category
    mycursor.execute("""
      CREATE TABLE IF NOT EXISTS
      category(
        category_id int NOT NULL PRIMARY KEY AUTO_INCREMENT
COMMENT 'Primary Key',
        create_time DATETIME COMMENT 'Create Time',
        update_time DATETIME COMMENT 'Update Time',
```

```
        category_name VARCHAR(255)
      );
      """)


    #payment_mode
    mycursor.execute("""
      CREATE TABLE IF NOT EXISTS
      payment_mode(
        payment_mode_id int NOT NULL PRIMARY KEY AUTO_INCREMENT
COMMENT 'Primary Key',
        create_time DATETIME COMMENT 'Create Time',
        update_time DATETIME COMMENT 'Update Time',
        payment_mode_name VARCHAR(255)
      );
      """)


    #menu
    mycursor.execute("""
      CREATE TABLE IF NOT EXISTS
      menu(
        menu_id int NOT NULL PRIMARY KEY AUTO_INCREMENT COMMENT
'Primary Key',
        create_time DATETIME COMMENT 'Create Time',
        update_time DATETIME COMMENT 'Update Time',
        item VARCHAR(255),
        price FLOAT,
        image VARCHAR(255),
        size_id INT, INDEX size_id (size_id),CONSTRAINT fk_size_id FOREIGN
KEY (size_id) REFERENCES size(size_id)  ,
        description VARCHAR(255),
        spice_lvl INT,
        is_veg BOOLEAN,
```

```
            category_id INT, INDEX category_id (category_id),CONSTRAINT
fk_category_id FOREIGN KEY (category_id) REFERENCES category(category_id)
        );
        """)


    #cafe_table
    mycursor.execute("""
        CREATE TABLE IF NOT EXISTS
        cafe_table(
            cafe_table_id int NOT NULL PRIMARY KEY AUTO_INCREMENT
COMMENT 'Primary Key',
            create_time DATETIME COMMENT 'Create Time',
            update_time DATETIME COMMENT 'Update Time',
            no_of_seats INT,
            cafe_table_number VARCHAR(255),
            location VARCHAR(255)
        );
        """)


    #customer
    mycursor.execute("""
        CREATE TABLE IF NOT EXISTS
        customer(
            customer_id int NOT NULL PRIMARY KEY AUTO_INCREMENT
COMMENT 'Primary Key',
            create_time DATETIME COMMENT 'Create Time',
            update_time DATETIME COMMENT 'Update Time',
            customer_name VARCHAR(255),
            customer_phone_number VARCHAR(255),
            gender BOOLEAN,
            email_id VARCHAR(255),
            DOB DATE,
            DOA DATE
```

```
        );
        """)


    #reservation
    mycursor.execute("""
        CREATE TABLE IF NOT EXISTS
        reservation(
            reservation_id int NOT NULL PRIMARY KEY AUTO_INCREMENT
COMMENT 'Primary Key',
            create_time DATETIME COMMENT 'Create Time',
            update_time DATETIME COMMENT 'Update Time',
            no_of_seats INT,
            date_time DATETIME,
            customer_id INT, INDEX customer_id (customer_id),CONSTRAINT
fk_customer_id FOREIGN KEY (customer_id) REFERENCES
customer(customer_id) ,
            cafe_table_id INT, INDEX cafe_table_id (cafe_table_id),CONSTRAINT
fk_cafe_table_id FOREIGN KEY (cafe_table_id) REFERENCES
cafe_table(cafe_table_id)
        );
        """)


    #cafe_order_header
    mycursor.execute("""
        CREATE TABLE IF NOT EXISTS
        cafe_order_header(
            cafe_order_header_id int NOT NULL PRIMARY KEY AUTO_INCREMENT
COMMENT 'Primary Key',
            create_time DATETIME COMMENT 'Create Time',
            update_time DATETIME COMMENT 'Update Time',
            cafe_table_id INT, INDEX cafe_table_id(cafe_table_id),CONSTRAINT
fk_cafe_order_header_cafe_table_id FOREIGN KEY (cafe_table_id) REFERENCES
cafe_table(cafe_table_id) ,
            total_amount FLOAT
```

```
        );
        """)


    #cafe_order_detail
    mycursor.execute("""
        CREATE TABLE IF NOT EXISTS
        cafe_order_detail(
            cafe_order_detail_id int NOT NULL PRIMARY KEY AUTO_INCREMENT
COMMENT 'Primary Key',
            cafe_order_header_id INT, INDEX
cafe_order_header_id(cafe_order_header_id),CONSTRAINT
fk_cafe_order_header_id FOREIGN KEY (cafe_order_header_id) REFERENCES
cafe_order_header(cafe_order_header_id) ,
            create_time DATETIME COMMENT 'Create Time',
            update_time DATETIME COMMENT 'Update Time',
            menu_id INT, INDEX menu_id(menu_id),CONSTRAINT
fk_cafe_order_detail_menu_id FOREIGN KEY (menu_id) REFERENCES
menu(menu_id) ,
            qty INT,
            unit_price float,
            gst float,
            amount FLOAT
        );
        """)


    #bill
    mycursor.execute("""
        CREATE TABLE IF NOT EXISTS
        bill(
            bill_id int NOT NULL PRIMARY KEY AUTO_INCREMENT COMMENT
'Primary Key',
            cafe_order_header_id INT, INDEX
cafe_order_header_id(cafe_order_header_id),CONSTRAINT
fk_cafe_order_header_id_bill FOREIGN KEY (cafe_order_header_id)
REFERENCES cafe_order_header(cafe_order_header_id) ,
```

```
            discount float,
            final_amount float,
            create_time DATETIME COMMENT 'Create Time',
            update_time DATETIME COMMENT 'Update Time',
            customer_id INT, INDEX customer_id (customer_id),CONSTRAINT
fk_bill_customer_id FOREIGN KEY (customer_id) REFERENCES
customer(customer_id) ,
            payment_mode_id INT, INDEX payment_mode_id
(payment_mode_id),CONSTRAINT fk_payment_mode_id FOREIGN KEY
(payment_mode_id) REFERENCES payment_mode(payment_mode_id) ,
            date_time DATETIME
          );
          """)
        mydb.commit()


    def seed(self):
        mydb = self.connect()
        mycursor = mydb.cursor()
        #INSERT DEFAULT RECORDS
        #size
        mycursor.execute("insert into size(size_name,create_time,update_time)
VALUES('Demi',now(),now());")
        mycursor.execute("insert into size(size_name,create_time,update_time)
VALUES('Short',now(),now());")
        mycursor.execute("insert into size(size_name,create_time,update_time)
VALUES('Tall',now(),now());")
        mycursor.execute("insert into size(size_name,create_time,update_time)
VALUES('Venti',now(),now());")
        mycursor.execute("insert into size(size_name,create_time,update_time)
VALUES('Grande',now(),now());")
        mycursor.execute("insert into size(size_name,create_time,update_time)
VALUES('Trenta',now(),now());")


        mydb.commit()
```

```
#category

mycursor.execute("insert into
category(category_name,create_time,update_time) VALUES('Hot
Beverages',now(),now());")

mycursor.execute("insert into
category(category_name,create_time,update_time) VALUES('Cold
Beverages',now(),now());")

mycursor.execute("insert into
category(category_name,create_time,update_time) VALUES('Quick
Bites',now(),now());")

mycursor.execute("insert into
category(category_name,create_time,update_time)
VALUES('Sandwiches',now(),now());")

mycursor.execute("insert into
category(category_name,create_time,update_time)
VALUES('Desserts',now(),now());")

mycursor.execute("insert into
category(category_name,create_time,update_time)
VALUES('Donuts',now(),now());")

mycursor.execute("insert into
category(category_name,create_time,update_time) VALUES('English
Breakfast',now(),now());")

mycursor.execute("insert into
category(category_name,create_time,update_time)
VALUES('Soups',now(),now());")

mycursor.execute("insert into
category(category_name,create_time,update_time) VALUES('Grab-n-
Go',now(),now());")

mydb.commit()


#payment_mode

mycursor.execute("insert into
payment_mode(payment_mode_name,create_time,update_time)
VALUES('Cash',now(),now());")

mycursor.execute("insert into
payment_mode(payment_mode_name,create_time,update_time)
VALUES('Card',now(),now());")

mycursor.execute("insert into
payment_mode(payment_mode_name,create_time,update_time)
VALUES('UPI',now(),now());")
```

```
mydb.commit()


#menu
mycursor.execute("insert into menu(item, price, size_id, description, spice_lvl, is_veg, category_id, create_time,update_time) VALUES('Cappucino', 399, null, null, 0, True, 1, now(),now());")

mycursor.execute("insert into menu(item, price, size_id, description, spice_lvl, is_veg, category_id, create_time,update_time) VALUES('Yamamoto Tea', 299, null, null, 0, True, 1, now(),now());")

mycursor.execute("insert into menu(item, price, size_id, description, spice_lvl, is_veg, category_id, create_time,update_time) VALUES('Miguel Drink', 249, null, null, 0, True, 1, now(),now());")

mycursor.execute("insert into menu(item, price, size_id, description, spice_lvl, is_veg, category_id, create_time,update_time) VALUES('Hot Chocolate', 349, null, null, 0, True, 1, now(),now());")

mycursor.execute("insert into menu(item, price, size_id, description, spice_lvl, is_veg, category_id, create_time,update_time) VALUES('Frappe', 399, null, null, 0, True, 2, now(),now());")

mycursor.execute("insert into menu(item, price, size_id, description, spice_lvl, is_veg, category_id, create_time,update_time) VALUES('Iced Tea', 299, null, null, 0, True, 2, now(),now());")

mycursor.execute("insert into menu(item, price, size_id, description, spice_lvl, is_veg, category_id, create_time,update_time) VALUES('Oreo Milkshake', 249, null, null, 0, True, 2, now(),now());")

mycursor.execute("insert into menu(item, price, size_id, description, spice_lvl, is_veg, category_id, create_time,update_time) VALUES('Blue Lagoon Mojito', 199, null, null, 0, True, 2, now(),now());")

mycursor.execute("insert into menu(item, price, size_id, description, spice_lvl, is_veg, category_id, create_time,update_time) VALUES('Corn Cheese Balls', 349, null, null, 1, True, 3, now(),now());")

mycursor.execute("insert into menu(item, price, size_id, description, spice_lvl, is_veg, category_id, create_time,update_time) VALUES('Potato Wedges', 199, null, null, 1, True, 3, now(),now());")

mycursor.execute("insert into menu(item, price, size_id, description, spice_lvl, is_veg, category_id, create_time,update_time) VALUES('Peri Peri Fries', 299, null, null, 2, True, 3, now(),now());")

mycursor.execute("insert into menu(item, price, size_id, description, spice_lvl, is_veg, category_id, create_time,update_time) VALUES('Merry Mellow Chicken Bites', 249, null, null, 3, False, 3, now(),now());")
```

```
mycursor.execute("insert into menu(item, price, size_id, description, spice_lvl, is_veg, category_id, create_time,update_time) VALUES('White Italian Tomato Sandwich', 249, null, null, 1, True, 4, now(),now());")

mycursor.execute("insert into menu(item, price, size_id, description, spice_lvl, is_veg, category_id, create_time,update_time) VALUES('Mexican Grilled Cheese Sandwich', 299, null, null, 2, True, 4, now(),now());")

mycursor.execute("insert into menu(item, price, size_id, description, spice_lvl, is_veg, category_id, create_time,update_time) VALUES('Chicken Tikka Sandwich', 249, null, null, 3, False, 4, now(),now());")

mycursor.execute("insert into menu(item, price, size_id, description, spice_lvl, is_veg, category_id, create_time,update_time) VALUES('Mutton Sheekh Sandwich', 299, null, null, 3, False, 4, now(),now());")

mycursor.execute("insert into menu(item, price, size_id, description, spice_lvl, is_veg, category_id, create_time,update_time) VALUES('Tiramisu', 249, null, null, 0, True, 5, now(),now());")

mycursor.execute("insert into menu(item, price, size_id, description, spice_lvl, is_veg, category_id, create_time,update_time) VALUES('Sizzling Chocolate Brownie', 299, null, null, 0, True, 5, now(),now());")

mycursor.execute("insert into menu(item, price, size_id, description, spice_lvl, is_veg, category_id, create_time,update_time) VALUES('Cheesecake', 149, null, null, 0, True, 5, now(),now());")

mycursor.execute("insert into menu(item, price, size_id, description, spice_lvl, is_veg, category_id, create_time,update_time) VALUES('Gelato Ice Cream', 149, null, null, 0, True, 5, now(),now());")

mycursor.execute("insert into menu(item, price, size_id, description, spice_lvl, is_veg, category_id, create_time,update_time) VALUES('Oreo Cookies and Creme Donut', 149, null, null, 0, True, 6, now(),now());")

mycursor.execute("insert into menu(item, price, size_id, description, spice_lvl, is_veg, category_id, create_time,update_time) VALUES('Glazed Blueberry Donut', 249, null, null, 0, True, 6, now(),now());")

mycursor.execute("insert into menu(item, price, size_id, description, spice_lvl, is_veg, category_id, create_time,update_time) VALUES('Reeses Classic Donut', 249, null, null, 0, True, 6, now(),now());")

mycursor.execute("insert into menu(item, price, size_id, description, spice_lvl, is_veg, category_id, create_time,update_time) VALUES('Chocolate Iced Raspberry Donut', 149, null, null, 0, True, 6, now(),now());")

mycursor.execute("insert into menu(item, price, size_id, description, spice_lvl, is_veg, category_id, create_time,update_time) VALUES('Croissant', 99, null, null, 0, True, 7, now(),now());")

mycursor.execute("insert into menu(item, price, size_id, description, spice_lvl, is_veg, category_id, create_time,update_time) VALUES('Bagel', 99, null, null, 0, True, 7, now(),now());")
```

```
mycursor.execute("insert into menu(item, price, size_id, description, spice_lvl, is_veg, category_id, create_time,update_time) VALUES('Sunny side up', 149, null, null, 1, False, 7, now(),now());")

mycursor.execute("insert into menu(item, price, size_id, description, spice_lvl, is_veg, category_id, create_time,update_time) VALUES('Bacon and Mushroom Peas', 249, null, null, 1, False, 7, now(),now());")

mycursor.execute("insert into menu(item, price, size_id, description, spice_lvl, is_veg, category_id, create_time,update_time) VALUES('Hot N Sour Soup', 99, null, null, 1, True, 8, now(),now());")

mycursor.execute("insert into menu(item, price, size_id, description, spice_lvl, is_veg, category_id, create_time,update_time) VALUES('Sweet Corn Soup', 99, null, null, 1, True, 8, now(),now());")

mycursor.execute("insert into menu(item, price, size_id, description, spice_lvl, is_veg, category_id, create_time,update_time) VALUES('Cream of Mushroom Soup', 99, null, null, 0, True, 8, now(),now());")

mycursor.execute("insert into menu(item, price, size_id, description, spice_lvl, is_veg, category_id, create_time,update_time) VALUES('Clear Chicken Soup', 99, null, null, 1, False, 8, now(),now());")

mycursor.execute("insert into menu(item, price, size_id, description, spice_lvl, is_veg, category_id, create_time,update_time) VALUES('Veg Frankie', 199, null, null, 1, True, 9, now(),now());")

mycursor.execute("insert into menu(item, price, size_id, description, spice_lvl, is_veg, category_id, create_time,update_time) VALUES('Chicken Frankie', 199, null, null, 2, False, 9, now(),now());")

mycursor.execute("insert into menu(item, price, size_id, description, spice_lvl, is_veg, category_id, create_time,update_time) VALUES('Choco Chip Cookies', 99, null, null, 0, True, 9, now(),now());")

mycursor.execute("insert into menu(item, price, size_id, description, spice_lvl, is_veg, category_id, create_time,update_time) VALUES('Spicy Chips', 99, null, null, 2, True, 9, now(),now());")

mydb.commit()


#cafe_table

mycursor.execute("insert into cafe_table(cafe_table_number, no_of_seats, location, create_time,update_time) VALUES('Table 01', 2, '1st Floor, North East', now(),now());")

mycursor.execute("insert into cafe_table(cafe_table_number, no_of_seats, location, create_time,update_time) VALUES('Table 02', 4, '1st Floor, South', now(),now());")
```

```
        mycursor.execute("insert into cafe_table(cafe_table_number, no_of_seats,
location, create_time,update_time) VALUES('Table 03', 6, '1st Floor, Center',
now(),now());")

        mycursor.execute("insert into cafe_table(cafe_table_number, no_of_seats,
location, create_time,update_time) VALUES('Table 04', 2, '2nd Floor, North East',
now(),now());")

        mycursor.execute("insert into cafe_table(cafe_table_number, no_of_seats,
location, create_time,update_time) VALUES('Table 05', 4, '2nd Floor, South',
now(),now());")

        mycursor.execute("insert into cafe_table(cafe_table_number, no_of_seats,
location, create_time,update_time) VALUES('Table 06', 6, '2nd Floor, Center',
now(),now());")

        mycursor.execute("insert into cafe_table(cafe_table_number, no_of_seats,
location, create_time,update_time) VALUES('Table 07', 2, '3rd Floor, North East',
now(),now());")

        mycursor.execute("insert into cafe_table(cafe_table_number, no_of_seats,
location, create_time,update_time) VALUES('Table 08', 4, '3rd Floor, South',
now(),now());")

        mycursor.execute("insert into cafe_table(cafe_table_number, no_of_seats,
location, create_time,update_time) VALUES('Table 09', 6, '3rd Floor, Center',
now(),now());")

        mydb.commit()


    def customer_create(self, cust=Customer):

        mydb = self.connect()

        mycursor = mydb.cursor()


        sql = "insert into customer(customer_name,customer_phone_number,
gender, email_id, DOB, DOA, create_time,update_time) VALUES (%s, %s, %s, %s,
%s, %s, now(), now())"

        val = (cust.customer_name, cust.phone_number, cust.gender, cust.email_id,
cust.DOB, cust.anniversary_date)


        mycursor.execute(sql, val)


        mydb.commit()

        print('customer created ')
```

```
    def customer_update(self, cust=Customer):
        mydb = self.connect()
        mycursor = mydb.cursor()

        sql = "update customer set
customer_name=%s,customer_phone_number=%s, gender=%s, email_id=%s,
DOB=%s, DOA=%s, update_time=now() where customer_id=%s"
        val = (cust.customer_name, cust.phone_number, cust.gender, cust.email_id,
cust.DOB, cust.anniversary_date, cust.customer_id)

        mycursor.execute(sql, val)

        mydb.commit()
        print('customer updated')

    def customer_delete(self, customer_id):
        mydb = self.connect()
        mycursor = mydb.cursor()

        sql = "delete from customer where customer_id=" + str(customer_id)

        mycursor.execute(sql)

        mydb.commit()
        print('customer updated')

    def bill_delete(self, bill_id):
        mydb = self.connect()
        mycursor = mydb.cursor()

        sql = "delete from bill where bill_id=" + str(bill_id)

        mycursor.execute(sql)
```

```
        mydb.commit()
        print('bill updated')


    def order_delete(self, cafe_order_header_id):
        mydb = self.connect()
        mycursor = mydb.cursor()


        sql = "delete from cafe_order_detail where cafe_order_header_id=" +
str(cafe_order_header_id)


        mycursor.execute(sql)


        sql = "delete from cafe_order_header where cafe_order_header_id=" +
str(cafe_order_header_id)


        mycursor.execute(sql)


        mydb.commit()
        print('order updated')


    def customer_get_all(self):
        mydb = self.connect()
        mycursor = mydb.cursor()
        mycursor.execute("select customer_id,
customer_name,customer_phone_number, gender, email_id, DOB, DOA,
create_time,update_time from customer order by customer_id desc")


        list_of_customer = mycursor.fetchall()
        cust_list = []
        for data in list_of_customer:
            cust = Customer()
            cust.customer_id = data[0]
```

```
            cust.customer_name = data[1]
            cust.phone_number = data[2]
            cust.gender = data[3]
            cust.email_id = data[4]
            cust.DOB = data[5]
            cust.anniversary_date = data[6]
            cust_list.append(cust)
            print(f"|{cust.customer_id:4}")

        return cust_list

    def customer_get_by_id(self, customer_id):
        mydb = self.connect()
        mycursor = mydb.cursor()
        mycursor.execute("select customer_id,
customer_name,customer_phone_number, gender, email_id, DOB, DOA,
create_time,update_time from customer where customer_id=" +
str(customer_id))

        list_of_customer = mycursor.fetchall()
        cust_list = []
        for data in list_of_customer:
            cust = Customer()
            cust.customer_id = data[0]
            cust.customer_name = data[1]
            cust.phone_number = data[2]
            cust.gender = data[3]
            cust.email_id = data[4]
            cust.DOB = data[5]
            cust.anniversary_date = data[6]
            cust_list.append(cust)
            print(f"|{cust.customer_id:4}")
```

```
        return cust_list


    def order_header_create(self, ord=OrderHeader):
        mydb = self.connect()
        mycursor = mydb.cursor()

        sql = "insert into cafe_order_header(cafe_table_id, total_amount,
create_time,update_time) VALUES (%s, %s, now(), now())"
        val = (ord.table_id, ord.total_amount)

        mycursor.execute(sql, val)

        mydb.commit()
        print('Order created ')
        return mycursor.lastrowid

    def order_header_update(self, ord=OrderHeader):
        mydb = self.connect()
        mycursor = mydb.cursor()

        sql = "update cafe_order_header set total_amount = %s, update_time=now()
where cafe_order_header_id=%s"
        val = (ord.total_amount, ord.order_header_id)

        mycursor.execute(sql, val)

        mydb.commit()
        print('Order updated')
        return mycursor.lastrowid

    def order_detail_update(self, ord=OrderDetail):
        mydb = self.connect()
```

```
        mycursor = mydb.cursor()


        sql = "update cafe_order_detail set menu_id=%s, qty=%s, unit_price=%s,
gst=%s, amount=%s, update_time=now() where cafe_order_detail_id = %s"
        val = (ord.item_id, ord.qty, ord.price, ord.gst, ord.amount,
ord.order_detail_id)


        mycursor.execute(sql, val)


        mydb.commit()
        print('Order detail updated')


    def order_detail_create(self, ord=OrderDetail):
        mydb = self.connect()
        mycursor = mydb.cursor()


        sql = "insert into cafe_order_detail(cafe_order_header_id, menu_id, qty,
unit_price, gst, amount, create_time,update_time) VALUES (%s, %s, %s, %s, %s,
%s, now(), now())"
        val = (ord.order_header_id, ord.item_id, ord.qty, ord.price, ord.gst,
ord.amount)


        mycursor.execute(sql, val)


        mydb.commit()
        print('Order created ')


    def order_header_get_all(self):
        mydb = self.connect()
        mycursor = mydb.cursor()
        mycursor.execute("select cafe_order_header_id, c.cafe_table_id,
cafe_table_number, total_amount, c.create_time from cafe_order_header c inner
join cafe_table  t on c.cafe_table_id = t.cafe_table_id order by
cafe_order_header_id desc")
```

```
        list = mycursor.fetchall()
        output_list = []
        for data in list:
            ord = OrderHeader()
            ord.order_header_id = data[0]
            ord.table_id = data[1]
            ord.table_number = data[2]
            ord.total_amount = data[3]
            ord.create_time = data[4]

            output_list.append(ord)
            print(f"|{ord.order_header_id:4}")

        return output_list

    def order_header_get_upaid(self):
        mydb = self.connect()
        mycursor = mydb.cursor()
        mycursor.execute("""select c.cafe_order_header_id, c.cafe_table_id,
cafe_table_number, total_amount, c.create_time
                from cafe_order_header c
                left outer join bill b on c.cafe_order_header_id = b.cafe_order_header_id
                inner join cafe_table  t on c.cafe_table_id = t.cafe_table_id
                where b.bill_id is null
                order by cafe_order_header_id desc""")

        list = mycursor.fetchall()
        output_list = []
        for data in list:
            ord = OrderHeader()
            ord.order_header_id = data[0]
```

```
            ord.table_id = data[1]
            ord.table_number = data[2]
            ord.total_amount = data[3]
            ord.create_time = data[4]

            output_list.append(ord)
            print(f"|{ord.order_header_id:4}")

        return output_list

    def order_header_get_by_id(self, cafe_order_header_id):
        mydb = self.connect()
        mycursor = mydb.cursor()

        sql = "select cafe_order_header_id, c.cafe_table_id, cafe_table_number,
total_amount, c.create_time from cafe_order_header c inner join cafe_table  t on
c.cafe_table_id = t.cafe_table_id where c.cafe_order_header_id = '" +
str(cafe_order_header_id) + "' order by cafe_order_header_id desc"

        mycursor.execute(sql)

        list = mycursor.fetchall()
        output_list = []
        for data in list:
            ord = OrderHeader()
            ord.order_header_id = data[0]
            ord.table_id = data[1]
            ord.table_number = data[2]
            ord.total_amount = data[3]
            ord.create_time = data[4]

            output_list.append(ord)
            print(f"|{ord.order_header_id:4}")
```

```
        return output_list


    def order_detail_get_all(self):
        mydb = self.connect()
        mycursor = mydb.cursor()
        mycursor.execute("select cafe_order_detail_id, cafe_order_header_id,
c.cafe_table_id, cafe_table_number, c.menu_id, item, qty, unit_price, gst, amount,
c.create_time from cafe_order_detail c inner join menu m on c.menu_id =
m.menu_id inner join cafe_table  t on c.cafe_table_id = t.cafe_table_id order by
cafe_order_detail_id")

        list = mycursor.fetchall()
        output_list = []
        for data in list:
            ord = OrderDetail()
            ord.cafe_order_detail_id = data[0]
            ord.cafe_order_header_id = data[1]
            ord.table_id = data[2]
            ord.table_number = data[3]
            ord.item_id = data[4]
            ord.item = data[5]
            ord.qty = data[6]
            ord.price = data[7]
            ord.gst = data[8]
            ord.amount = data[9]
            ord.create_time = data[10]

            output_list.append(ord)
            print(f"|{ord.cafe_order_detail_id:4}")

        return output_list
```

```
    def order_detail_get_by_id(self, cafe_order_header_id):
        mydb = self.connect()
        mycursor = mydb.cursor()

        sql = "select c.cafe_order_detail_id, c.cafe_order_header_id, c.menu_id, item,
qty, unit_price, gst, amount, c.create_time from cafe_order_detail c inner join
menu m on c.menu_id = m.menu_id where c.cafe_order_header_id = '" +
str(cafe_order_header_id) + "' order by c.cafe_order_detail_id"

        mycursor.execute(sql)

        list = mycursor.fetchall()
        output_list = []
        for data in list:
            ord = OrderDetail()
            ord.cafe_order_detail_id = data[0]
            ord.cafe_order_header_id = data[1]
            ord.item_id = data[2]
            ord.item = data[3]
            ord.qty = data[4]
            ord.price = data[5]
            ord.gst = data[6]
            ord.amount = data[7]
            ord.create_time = data[8]

            output_list.append(ord)
            print(f"|{ord.cafe_order_detail_id:4}")

        return output_list

    def order_detail_delete_by_id(self, cafe_order_header_id):
        mydb = self.connect()
        mycursor = mydb.cursor()
```

```
        sql = "delete from cafe_order_detail where cafe_order_header_id = " +
str(cafe_order_header_id)


        mycursor.execute(sql)
        mydb.commit()
        print('Order detail deleted:', cafe_order_header_id)


    def item_get_all(self):
        mydb = self.connect()
        mycursor = mydb.cursor()
        mycursor.execute("select menu_id, item, price, description, spice_lvl, is_veg,
m.category_id, category_name from menu m inner join category c on
m.category_id = c.category_id")


        list = mycursor.fetchall()
        output_list = []
        for data in list:
            itm = Item()
            itm.item_id = data[0]
            itm.item = data[1]
            itm.price = data[2]
            itm.description = data[3]
            itm.spice_level = data[4]
            itm.veg = data[5]
            itm.category_id = data[6]
            itm.category = data[7]


            output_list.append(itm)
            print(f"|{itm.item_id:4}")


        return output_list
```

```
    def item_get_by_category_id(self, category_id):
        mydb = self.connect()
        mycursor = mydb.cursor()
        mycursor.execute("select menu_id, item, price, description, spice_lvl, is_veg,
m.category_id, category_name from menu m inner join category c on
m.category_id = c.category_id where m.category_id=" + str(category_id))

        list = mycursor.fetchall()
        output_list = []
        for data in list:
            itm = Item()
            itm.item_id = data[0]
            itm.item = data[1]
            itm.price = data[2]
            itm.description = data[3]
            itm.spice_level = data[4]
            itm.veg = data[5]
            itm.category_id = data[6]
            itm.category = data[7]

            output_list.append(itm)
            print(f"|{itm.item_id:4}")

        return output_list

    def table_get_all(self):
        mydb = self.connect()
        mycursor = mydb.cursor()
        mycursor.execute("select cafe_table_id, cafe_table_number, no_of_seats,
location from cafe_table")

        list = mycursor.fetchall()
        output_list = []
```

```
        for data in list:
            itm = Table()
            itm.table_id = data[0]
            itm.table_number = data[1]
            itm.number_of_seats = data[2]
            itm.location = data[3]

            output_list.append(itm)
            print(f"|{itm.table_id:4}")

        return output_list

    def table_reservation_get_all(self):
        mydb = self.connect()
        mycursor = mydb.cursor()
        mycursor.execute("select reservation_id, r.no_of_seats, date_time,
r.customer_id, customer_name, r.cafe_table_id, cafe_table_number from
reservation r inner join customer c on r.customer_id = c.customer_id inner join
cafe_table ct on r.cafe_table_id = ct.cafe_table_id order by reservation_id desc")

        list = mycursor.fetchall()
        output_list = []
        for data in list:
            itm = TableReservation()
            itm.reservation_id = data[0]
            itm.pax = data[1]
            itm.datetime = data[2]
            itm.customer_id = data[3]
            itm.customer = data[4]
            itm.table_id = data[5]
            itm.table_number = data[6]

            output_list.append(itm)
```

```
            print(f"|{itm.reservation_id:4}")

        return output_list

    def table_reservation_get_by_id(self, reservation_id):
        mydb = self.connect()
        mycursor = mydb.cursor()
        mycursor.execute("select reservation_id, r.no_of_seats, date_time,
r.customer_id, customer_name, r.cafe_table_id, cafe_table_number from
reservation r inner join customer c on r.customer_id = c.customer_id inner join
cafe_table ct on r.cafe_table_id = ct.cafe_table_id where reservation_id = " +
str(reservation_id))

        list = mycursor.fetchall()
        output_list = []
        for data in list:
            itm = TableReservation()
            itm.reservation_id = data[0]
            itm.pax = data[1]
            itm.datetime = data[2]
            itm.customer_id = data[3]
            itm.customer = data[4]
            itm.table_id = data[5]
            itm.table_number = data[6]

            output_list.append(itm)
            print(f"|{itm.reservation_id:4}")

        return output_list

    def table_reservation_create(self, reserv=TableReservation):
        mydb = self.connect()
        mycursor = mydb.cursor()
```

```
        sql = "insert into reservation(no_of_seats,date_time, customer_id,
cafe_table_id, create_time,update_time) VALUES (%s, %s, %s, %s, now(),
now())"
        val = (reserv.pax, reserv.datetime, reserv.customer_id, reserv.table_id)

        mycursor.execute(sql, val)

        mydb.commit()
        print('Table Reservation created ')

    def table_reservation_update(self, reserv=TableReservation):
        mydb = self.connect()
        mycursor = mydb.cursor()

        sql = "update reservation set no_of_seats=%s,date_time=%s,
customer_id=%s, cafe_table_id=%s, update_time=now() where
reservation_id=%s"
        val = (reserv.pax, reserv.datetime, reserv.customer_id, reserv.table_id,
reserv.reservation_id)

        mycursor.execute(sql, val)

        mydb.commit()
        print('Table Reservation created ')

    def table_reservation_delete(self, reservation_id):
        mydb = self.connect()
        mycursor = mydb.cursor()

        sql = "delete from reservation where reservation_id=" + str(reservation_id)

        mycursor.execute(sql)
```

```
        mydb.commit()
        print('Table Reservation created ')


    def payment_mode_get_all(self):
        mydb = self.connect()
        mycursor = mydb.cursor()
        mycursor.execute("select payment_mode_id, payment_mode_name from
payment_mode")


        list = mycursor.fetchall()
        output_list = []
        for data in list:
            itm = PaymentMode()
            itm.payment_mode_id = data[0]
            itm.payment_mode_name = data[1]


            output_list.append(itm)
            print(f"|{itm.payment_mode_id:4}")


        return output_list


    def bill_create(self, bill=Bill):
        mydb = self.connect()
        mycursor = mydb.cursor()


        sql = "insert into
bill(cafe_order_header_id,discount,final_amount,customer_id,
payment_mode_id, date_time, create_time,update_time) VALUES (%s, %s, %s,
%s, %s, %s, now(), now())"
        val = (bill.cafe_order_header_id,
bill.discount,bill.final_amount,bill.customer_id, bill.payment_mode_id,
bill.datetime)
```

```
        mycursor.execute(sql, val)


        mydb.commit()
        print('Bill created ')


    def bill_update(self, bill=Bill):
        mydb = self.connect()
        mycursor = mydb.cursor()


        sql = "update bill set customer_id=%s,
payment_mode_id=%s,discount=%s,final_amount=%s, update_time=now()
where bill_id=%s"
        val = (bill.customer_id, bill.payment_mode_id,
bill.discount,bill.final_amount,bill.bill_id)


        mycursor.execute(sql, val)


        mydb.commit()
        print('Bill created ')


    def bill_get_all(self):
        mydb = self.connect()
        mycursor = mydb.cursor()
        mycursor.execute("""select b.bill_id, b.cafe_order_header_id, b.customer_id,
c1.customer_name, b.payment_mode_id,
p.payment_mode_name,b.discount,b.final_amount,b.date_time, c.cafe_table_id,
c.cafe_table_number, ch.total_amount
            from bill b
            inner join cafe_order_header ch on b.cafe_order_header_id =
ch.cafe_order_header_id
            inner join cafe_table c on ch.cafe_table_id = c.cafe_table_id
            inner join customer c1 on b.customer_id = c1.customer_id
            inner join payment_mode p on b.payment_mode_id =
p.payment_mode_id
```

```
            order by b.bill_id desc
            """)

        list = mycursor.fetchall()
        output_list = []
        for data in list:
            bill = Bill()
            bill.bill_id = data[0]
            bill.cafe_order_header_id = data[1]
            bill.customer_id = data[2]
            bill.customer = data[3]
            bill.payment_mode_id = data[4]
            bill.mode_of_payment = data[5]
            bill.discount = data[6]
            bill.final_amount = data[7]
            bill.datetime = data[8]
            bill.table_id = data[9]
            bill.table_number = data[10]
            bill.total_amount = data[11]

            output_list.append(bill)
            print(f"|{bill.bill_id:4}")

        return output_list

    def bill_get_by_id(self, bill_id):
        mydb = self.connect()
        mycursor = mydb.cursor()
        mycursor.execute("""select b.bill_id, b.cafe_order_header_id, b.customer_id,
c1.customer_name, b.payment_mode_id,
p.payment_mode_name,b.discount,b.final_amount, b.date_time, c.cafe_table_id,
c.cafe_table_number, ch.total_amount
            from bill b
```

```
            inner join cafe_order_header ch on b.cafe_order_header_id =
ch.cafe_order_header_id
            inner join cafe_table c on ch.cafe_table_id = c.cafe_table_id
            inner join customer c1 on b.customer_id = c1.customer_id
            inner join payment_mode p on b.payment_mode_id =
p.payment_mode_id
            where b.bill_id = """ + str(bill_id) +
            """ order by b.bill_id desc""")

        list = mycursor.fetchall()
        output_list = []
        for data in list:
            bill = Bill()
            bill.bill_id = data[0]
            bill.cafe_order_header_id = data[1]
            bill.customer_id = data[2]
            bill.customer = data[3]
            bill.payment_mode_id = data[4]
            bill.mode_of_payment = data[5]
            bill.discount = data[6]
            bill.final_amount = data[7]
            bill.datetime = data[8]
            bill.table_id = data[9]
            bill.table_number = data[10]
            bill.total_amount = data[11]

            output_list.append(bill)
            print(f"|{bill.bill_id:4}")

        return output_list

    def bill_get_buy_order_header_id(self,cafe_order_header_id):
```

```
        mydb = self.connect()
        mycursor = mydb.cursor()

        sql = "select bill_id from bill where cafe_order_header_id =
"+str(cafe_order_header_id)
        mycursor.execute(sql)
        list = mycursor.fetchall()
        output_list = []
        for data in list:
            bill = Bill()
            bill.bill_id = data[0]

            output_list.append(bill)
            print(f"|{bill.bill_id:4}")

        return output_list

    def bill_get_buy_customer_id(self,customer_id):
        mydb = self.connect()
        mycursor = mydb.cursor()

        sql = "select bill_id from bill where customer_id = "+str(customer_id)
        mycursor.execute(sql)
        list = mycursor.fetchall()
        output_list = []
        for data in list:
            bill = Bill()
            bill.bill_id = data[0]

            output_list.append(bill)
            print(f"|{bill.bill_id:4}")
```

```
        return output_list


    def category_get_all(self):
        mydb = self.connect()
        mycursor = mydb.cursor()
        mycursor.execute("""select category_id, category_name from category order
by category_id""")

        list = mycursor.fetchall()
        output_list = []
        for data in list:
            cat = Category()
            cat.category_id = data[0]
            cat.category_name = data[1]

            output_list.append(cat)
            print(f"|{cat.category_id:4}")

        return output_list
```

Bill_Create.py

```
import tkinter as tk
from tkinter import StringVar, messagebox

from datetime import datetime
from model.bill import Bill
from view.helper.comp_helper import ComponentHelper
from database.db import Database
from view.event import Event

class BillCreate():

    cafe_order_header_id=0
    id=0
    def __init__(self):
        print('create bill constructor')
```

```
        self.OnViewUpdated = Event()

    def ViewUpdated(self):
        self.OnViewUpdated()

    def AddSubscribersForViewUpdatedEvent(self,objMethod):
        self.OnViewUpdated += objMethod

    def RemoveSubscribersForViewUpdatedEvent(self,objMethod):
        self.OnViewUpdated -= objMethod

    def createWidgets(self, win, id=0):
        self.id = id
        top=win.winfo_toplevel()

        top.rowconfigure(0, weight=1)
        top.columnconfigure(0, weight=1)

        win.rowconfigure(1, weight=1)
        win.columnconfigure(1, weight=1)

        helper = ComponentHelper()
        win.frame = helper.add_background(win, "./images/bill_add_del.gif")

        db = Database()
        self.order_header_list = db.order_header_get_upaid()
        order_header_arr = []
        order_header_arr.append('Please Select')
        for data in self.order_header_list:
            order_header_arr.append(str(data.order_header_id) + "|" +
str(data.table_number) + "|" + str(data.total_amount) + "|" +
str(data.create_time))

        self.table_list = db.table_get_all()
        table_arr = []
        table_arr.append('Please Select')
        for data in self.table_list:
            table_arr.append(data.table_number)

        self.cust_list = db.customer_get_all()
        cust_arr = []
        cust_arr.append('Please Select')
        for data in self.cust_list:
            cust_arr.append(data.customer_name)

        self.payment_mode_list = db.payment_mode_get_all()
        payment_mode_arr = []
        payment_mode_arr.append('Please Select')
```

```
        for data in self.payment_mode_list:
            payment_mode_arr.append(data.payment_mode_name)

        bill = Bill()
        db = Database()
        if(id > 0):
            list = db.bill_get_by_id(id)
            for data in list:
                bill.bill_id = data.bill_id
                bill.cafe_order_header_id=data.cafe_order_header_id
                bill.payment_mode_id = data.payment_mode_id
                bill.mode_of_payment = data.mode_of_payment
                bill.datetime = data.datetime
                bill.customer_id = data.customer_id
                bill.customer = data.customer
                bill.table_id = data.table_id
                bill.table_number = data.table_number
                bill.total_amount=data.total_amount
                bill.discount = data.discount
                bill.final_amount = data.final_amount
            order_txt = ''
            for data in self.order_header_list:
                if(data.cafe_order_header_id == bill.cafe_order_header_id):
                    order_txt = str(data.order_header_id) + "|" +
str(data.table_number) + "|" + str(data.total_amount) + "|" +
str(data.create_time)


        helper = ComponentHelper()
        if(self.id > 0):
            self.order = helper.create_label_options_menu(win.frame,
0,'Order', order_header_arr, self.order_changed, order_txt)
            self.cust = helper.create_label_options_menu(win.frame,
4,'Customer', cust_arr, self.customer_changed, bill.customer)
            self.mode_of_payment = helper.create_label_options_menu(win.frame,
7,'Mode Of Payment', payment_mode_arr, self.item_changed,
bill.mode_of_payment)
        else:
            self.order = helper.create_label_options_menu(win.frame,
0,'Order', order_header_arr, self.order_changed)
            self.cust = helper.create_label_options_menu(win.frame,
4,'Customer', cust_arr, self.customer_changed)
            self.mode_of_payment = helper.create_label_options_menu(win.frame,
7,'Mode Of Payment', payment_mode_arr, self.item_changed)

        self.table = helper.create_label_label(win.frame, 1,'Table',
bill.table_number)
```

```
        self.total_amount = helper.create_label_label(win.frame, 2,'Total
Amount', bill.total_amount)
        self.datetime = helper.create_label_label(win.frame, 3, 'DateTime',
bill.datetime)

        self.sv = StringVar()
        self.discount = helper.create_label_entry(win.frame, 5,'Discount',
bill.discount, self.sv, self.discount_changed)
        self.final_amount = helper.create_label_label(win.frame, 6,'Final
Amount', bill.final_amount)

        cancel = tk.Button(win.frame, text='Cancel',
command=lambda:self.bill_cancel())
        cancel.grid(row=8, column=0, sticky=tk.N+tk.S+tk.E+tk.W)

        self.submit = tk.Button(win.frame, text='Submit',
command=lambda:self.bill_create())
        self.submit.grid(row=8, column=1, sticky=tk.N+tk.S+tk.E+tk.W)

        if(self.id > 0):
            self.order[1].configure(state="disable")

    def discount_changed(self):
        print(self.sv.get())

        final_amount = float(self.total_amount.cget("text")) * (1-
(float(self.discount.get())/100))
        self.final_amount.config(text=final_amount)

        return True

    def bill_cancel(self, *args):
        self.ViewUpdated()

    def order_changed(self, *args):
        args_arr = args[0].split('|')
        self.cafe_order_header_id = args_arr[0]
        self.table.config(text=args_arr[1])
        self.total_amount.config(text=args_arr[2])
        self.datetime.config(text=args_arr[3])
        final_amount = float(self.total_amount.cget("text")) * (1-
(float(self.discount.get())/100))
        self.final_amount.config(text=final_amount)
        print('data', self)

    def item_changed(self, *args):
        print(self)
```

```
    def customer_changed(self, *args):
        print(self)
        for data in self.cust_list:
            if(data.customer_name == args[0]):
                dob_month = datetime.strptime(str(data.DOB), '%Y-%m-%d').month
                dob_date = datetime.strptime(str(data.DOB), '%Y-%m-%d').day

                bill_month = datetime.strptime(self.datetime.cget("text"),
'%Y-%m-%d %H:%M:%S').month
                bill_date = datetime.strptime(self.datetime.cget("text"), '%Y-
%m-%d %H:%M:%S').day

                if(dob_month == bill_month and dob_date == bill_date):
                    messagebox.showinfo('Success!', 'Wish the Customer Happy
Birthday! Added 50% discount to the bill!')
                    helper = ComponentHelper()
                    helper.change_text(self.discount,'50')
                    final_amount = float(self.total_amount.cget("text")) *
(0.50)
                    self.final_amount.config(text=final_amount)
                    return

    def bill_create(self):
        if self.cust[0].get() == "Please Select":
            messagebox.showerror('Failure!', 'Please Select Customer!')
            return
        if self.mode_of_payment[0].get() == "Please Select":
            messagebox.showerror('Failure!', 'Please Select Mode of Payment!')
            return

        db = Database()

        bill = Bill()
        bill.cafe_order_header_id = self.cafe_order_header_id
        for data in self.cust_list:
            if data.customer_name == self.cust[0].get():
                bill.customer_id = data.customer_id

        for data in self.payment_mode_list:
            if data.payment_mode_name == self.mode_of_payment[0].get():
                bill.payment_mode_id = data.payment_mode_id

        bill.datetime = self.datetime.cget("text")
        bill.discount = self.discount.get()
        final_amount = float(self.total_amount.cget("text")) * (1-
(float(self.discount.get())/100))
        self.final_amount.config(text=final_amount)
        bill.final_amount = final_amount
```

```
        if(self.id > 0):
            bill.bill_id = self.id
            db.bill_update(bill)
        else:
            db.bill_create(bill)

        self.ViewUpdated()
        messagebox.showinfo('Success!', 'Bill Created Successfully')
```

Customer_create.py

```
from datetime import datetime, timedelta
import tkinter as tk
from tkinter import messagebox

from model.customer import Customer
from view.helper.comp_helper import ComponentHelper
from view.event import Event
from database.db import Database

import mysql.connector as mysql
import re

class CustomerCreate():
    name = tk.Entry
    phone = tk.Entry
    gender = tk.Entry
    email = tk.Entry
    dob = tk.Entry
    doa = tk.Entry
    table_list = []
    customer_list = []
    cust_id = 0

    def __init__(self, win):
        print('create customer constructor')
        self.OnViewUpdated = Event()

    def ViewUpdated(self):
        self.OnViewUpdated()

    def AddSubscribersForViewUpdatedEvent(self,objMethod):
        self.OnViewUpdated += objMethod
```

```
    def RemoveSubscribersForViewUpdatedEvent(self,objMethod):
        self.OnViewUpdated -= objMethod

    def createWidgets(self, win, cust_id=0):
        top=win.winfo_toplevel()

        top.rowconfigure(0, weight=1)
        top.columnconfigure(0, weight=1)

        win.rowconfigure(1, weight=1)
        win.columnconfigure(1, weight=1)

        helper = ComponentHelper()
        win.frame = helper.add_background(win,
"./images/customer_add_delete.gif")

        cust = Customer()
        cust.gender = "Please Select"
        db = Database()
        if(cust_id > 0):
            self.cust_id = cust_id
            cust_list = db.customer_get_by_id(cust_id)
            for data in cust_list:
                cust.customer_id = self.cust_id
                cust.customer_name = data.customer_name
                cust.phone_number = data.phone_number
                if data.gender == 1:
                    cust.gender = "Male"
                elif data.gender == 2:
                    cust.gender = "Female"
                else:
                    cust.gender = "Other"
                cust.gender = data.gender
                cust.email_id = data.email_id
                cust.DOB = data.DOB
                cust.anniversary_date = data.anniversary_date

        helper = ComponentHelper()
        self.name = helper.create_label_entry(win.frame, 1,'Customer Name',
cust.customer_name)
        self.phone = helper.create_label_entry(win.frame, 2,'Phone Number',
cust.phone_number)
        self.gender = helper.create_label_options_menu(win.frame, 3,'Gender',
['Male','Female','Other'], self.item_changed,cust.gender )
        self.email = helper.create_label_entry(win.frame, 4,'Email Id',
cust.email_id)
        self.dob = helper.create_label_entry(win.frame, 5,'Date Of Birth',
cust.DOB)
```

```
        self.doa = helper.create_label_entry(win.frame, 6,'Anniversary Date',
cust.anniversary_date)

        self.cancel = tk.Button(win.frame, text='Cancel',
command=lambda:self.customer_cancel())
        self.cancel.grid(row=7, column=0, sticky=tk.N+tk.S+tk.E+tk.W)

        self.submit = tk.Button(win.frame, text='Submit',
command=lambda:self.customer_create())
        self.submit.grid(row=7, column=1, sticky=tk.N+tk.S+tk.E+tk.W)

    def item_changed(self):
        pass

    def customer_cancel(self):
        self.ViewUpdated()

    def customer_create(self):
        if self.name.get() == "":
            messagebox.showerror('Failure!', 'Please Enter Customer Name!')
            return
        if self.phone.get() == "":
            messagebox.showerror('Failure!', 'Please Enter The Phone Number!')
            return
        if len(self.phone.get()) < 5:
            messagebox.showerror('Failure!', 'Phone Number should be atleast 5
digits long!')
            return
        if len(self.phone.get()) > 15:
            messagebox.showerror('Failure!', 'Phone Number cannot be longer
than 15 digits!')
            return
        if self.gender[0].get() == "" or self.gender[0].get() == "Please
Select":
            messagebox.showerror('Failure!', 'Please Enter Gender!')
            return
        if self.email.get() == "":
            messagebox.showerror('Failure!', 'Please Enter The Email ID!')
            return

        try:
            res = bool(datetime.strptime(self.dob.get().strip(), '%Y-%m-%d'))
        except BaseException as e:
            print(e)
            messagebox.showerror('Failure!', 'Please enter appropriate
Birthday in YYYY-mm-dd format')
            return
```

```
        if datetime.strptime(self.dob.get().strip(), '%Y-%m-%d') <
(datetime.now() - timedelta(days=365*150)):
            messagebox.showerror('Failure!', 'Age should not be more than 150
years')
            return

        if datetime.strptime(self.dob.get().strip(), '%Y-%m-%d') >
(datetime.now() - timedelta(days=365*18)):
            messagebox.showerror('Failure!', 'Age should be at least 18
years')
            return

        try:
            res = bool(datetime.strptime(self.doa.get().strip(), '%Y-%m-%d'))
        except:
            messagebox.showerror('Failure!', 'Please enter appropriate
Anniversary Date in YYYY-mm-dd format')
            return

        if datetime.strptime(self.doa.get().strip(), '%Y-%m-%d') <
(datetime.now() - timedelta(days=365*150)):
            messagebox.showerror('Failure!', 'Anniversary should not be more
than 150 years')
            return

        if datetime.strptime(self.doa.get().strip(), '%Y-%m-%d') <
(datetime.strptime(self.dob.get().strip(), '%Y-%m-%d') +
timedelta(days=365*18)):
            messagebox.showerror('Failure!', '18 Years gap should be there
between Anniversary and Date of Birth')
            return

        if datetime.strptime(self.dob.get().strip(), '%Y-%m-%d') >
datetime.now():
            messagebox.showerror('Failure!', 'Date of Birth cannot be a future
date')
            return

        if datetime.strptime(self.doa.get().strip(), '%Y-%m-%d') >
datetime.now():
            messagebox.showerror('Failure!', 'Anniversary Date cannot be a
future date')
            return

        regex = r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b'

        if(re.fullmatch(regex, self.email.get())):
            print("Valid Email")
```

```
        else:
            messagebox.showerror('Failure!', 'Please Enter valid Email ID!')
            return

        db = Database()

        cust = Customer()
        cust.customer_name = self.name.get()
        cust.phone_number = self.phone.get()
        if self.gender[0].get() == 'Male':
            cust.gender = 1
        elif self.gender[0].get() == "Female":
            cust.gender = 2
        else:
            cust.gender = 3

        cust.email_id = self.email.get()
        cust.DOB = self.dob.get()
        cust.anniversary_date = self.doa.get()

        if(int(self.cust_id) > 0):
            cust.customer_id = self.cust_id
            db.customer_update(cust)
            messagebox.showinfo('Success!', 'Customer Updated Successfully')
        else:
            db.customer_create(cust)
            messagebox.showinfo('Success!', 'Customer Created Successfully')

        self.ViewUpdated()
```

order_create.py

```
import tkinter as tk

from tkinter import *
from tkinter.ttk import *
from tkinter import messagebox
from model import order_detail

from model.item import Item
from model.order_header import OrderHeader
from model.order_detail import OrderDetail

from view.helper.comp_helper import ComponentHelper
from view.view import View
from view.event import Event
```

```
from database.db import Database

class OrderCreate():
    i=1
    item = tk.OptionMenu
    table = tk.OptionMenu
    qty = tk.Entry
    price = tk.Label
    gst = tk.Label
    #amount = tk.Label
    description = tk.Label
    spice_lvl = tk.Label
    veg = tk.Label
    category = tk.Label
    item_list = []
    table_list = []
    order_details = []
    popup = tk.Menu
    tv = Treeview
    order_header_id = 0
    win=object

    def __init__(self):
        print('create order constructor')
        self.item_list = []
        self.table_list = []
        self.order_details = []
        self.OnViewUpdated = Event()

    def ViewUpdated(self):
        self.OnViewUpdated()

    def AddSubscribersForViewUpdatedEvent(self,objMethod):
        self.OnViewUpdated += objMethod

    def RemoveSubscribersForViewUpdatedEvent(self,objMethod):
        self.OnViewUpdated -= objMethod

    def createWidgets(self, win, order_header_id=''):
        top=win.winfo_toplevel()

        top.rowconfigure(0, weight=1)
        top.columnconfigure(0, weight=1)

        win.rowconfigure(1, weight=1)
        win.columnconfigure(1, weight=1)
        self.win = win
```

```
        helper = ComponentHelper()
        win.frame = helper.add_background(win, "./images/order_add_del.gif",
0.90)

        db = Database()
        self.item_list = db.item_get_all()
        item_arr = []
        item_arr.append('Please Select')
        for data in self.item_list:
            item_arr.append(data.item)

        self.table_list = db.table_get_all()
        table_arr = []
        table_arr.append('Please Select')
        for data in self.table_list:
            table_arr.append(data.table_number)

        self.category_list = db.category_get_all()
        category_arr = []
        category_arr.append('Please Select')
        for data in self.category_list:
            category_arr.append(data.category_name)

        helper = ComponentHelper()

        order_header = OrderHeader()

        if(order_header_id != ''):
            self.order_header_id = order_header_id
            order_header_list = db.order_header_get_by_id(order_header_id)
            for data in order_header_list:
                order_header.order_header_id = order_header_id
                order_header.table_id = data.table_id
                order_header.table_number = data.table_number
                order_header.total_amount = data.total_amount

        header = Frame(win.frame)
        header['padding'] = 1
        header['width'] = 500
        header['height'] = 100
        header['borderwidth'] = 1
        header['relief'] = 'sunken'
        header.grid(columnspan=2,sticky=tk.N+tk.S+tk.E+tk.W)

        self.table = helper.create_label_options_menu(header, 0,'Table',
table_arr, self.table_changed, order_header.table_number)
```

```
        self.total_amount = helper.create_label_label(header, 1,'Total
Amount', order_header.total_amount)

        self.category = helper.create_label_options_menu(win.frame,
2,'Category', category_arr, self.category_changed)
        self.item = helper.create_label_options_menu(win.frame, 3,'Item Name',
item_arr, self.item_changed)
        self.qty = helper.create_label_entry(win.frame, 4,'Quantity', 1)
        self.price = helper.create_label_label(win.frame, 5,'Price', 0)
        self.description = helper.create_label_label(win.frame,
6,'Description', '')
        self.spice_lvl = helper.create_label_label(win.frame, 7,'Spice Level',
1)
        self.veg = helper.create_label_label(win.frame, 8,'Veg', 'Yes')
        self.gst = helper.create_label_label(win.frame, 9,'GST', '18')

        button_frame = tk.Frame(win.frame, bg='#CC8066')
        #button_frame.place(relx=0,rely=0,relwidth=0.75,relheight=0.75)
        button_frame.grid(columnspan=2, sticky=tk.S)

        self.add = tk.Button(button_frame, text='Add Item',
command=lambda:self.AddItem(tv, False))
        self.add.grid(row=11, column=1,
sticky=tk.N+tk.S+tk.E+tk.W,padx=10,pady=10)

        self.update = tk.Button(button_frame, text='Update',
command=lambda:self.UpdateItem(tv))
        self.update.grid(row=11, column=2,
sticky=tk.N+tk.S+tk.E+tk.W,padx=10,pady=10)

        self.cancel = tk.Button(button_frame, text='Cancel',
command=lambda:self.Cancel())
        self.cancel.grid(row=11, column=3,
sticky=tk.N+tk.S+tk.E+tk.W,padx=10,pady=10)

        self.enable_insert(True)

        help = tk.Label(button_frame, text='*Right click to Edit or Delete the
Item')
        help.grid(row=12, column=1,
sticky=tk.N+tk.S+tk.E+tk.W,padx=10,pady=10)

        tv = self.CreateUI(win.frame, 13)

        if(order_header_id != ''):
            order_detail_list = db.order_detail_get_by_id(order_header_id)
            for data in order_detail_list:
                order_detail = OrderDetail()
```

```
                order_detail.order_detail_id = data.order_detail_id
                order_detail.order_header_id = order_header_id
                order_detail.item_id = data.item_id
                order_detail.item = data.item
                order_detail.qty = data.qty
                order_detail.price = data.price
                order_detail.amount = data.amount
                self.order_details.append(order_detail)
                tv.insert("", 'end', iid=None, text=self.i, values=(data.item,
data.qty, data.price, data.gst, data.amount))
                self.i = self.i + 1

        submit_button_frame = tk.Frame(win.frame, bg='#CC8066')
        submit_button_frame.grid(columnspan=2)

        self.submit = tk.Button(submit_button_frame, text='Submit Order',
command=lambda:self.order_create())
        self.submit.grid(row=14, column=1,
sticky=tk.N+tk.S+tk.E+tk.W,padx=10,pady=10)

        self.cancel_submit = tk.Button(submit_button_frame, text='Cancel',
command=lambda:self.order_cancel())
        self.cancel_submit.grid(row=14, column=2,
sticky=tk.N+tk.S+tk.E+tk.W,padx=10,pady=10)

    def category_changed(self, *args):
        self.price.config(text=0)
        self.description.config(text='')
        self.spice_lvl.config(text=1)
        self.veg.config(text='Yes')
        self.gst.config(text=18)

        category_id=0
        for data in self.category_list:
            if data.category_name == self.category[0].get():
                category_id = data.category_id

        self.item[0].set('')
        self.item[1]['menu'].delete(0, 'end')
        self.item[1]['menu'].add_command(label='Please Select',
command=tk._setit(self.item[0], 'Please Select'))

        db = Database()
        self.item_list = db.item_get_by_category_id(category_id)
        item_arr = []
        item_arr.append('Please Select')
        for data in self.item_list:
            item_arr.append(data.item)
```

```
            #self.item[1]['menu'].add_command(label=data.item,
command=lambda:self.item_changed(data.item))
            self.item[1]['menu'].add_command(label=data.item,
command=tk._setit(self.item[0], data.item, self.item_changed))

    def order_cancel(self):
        self.ViewUpdated()

    def Cancel(self):
        self.enable_insert(True)

    def UpdateItem(self, tv):
        self.AddItem(tv, True)

    def enable_insert(self, flag):
        if flag:
            self.add["state"] = "normal"
            self.item[1].configure(state="normal")
            self.category[1].configure(state="normal")
            self.update["state"] = "disabled"
            self.cancel["state"] = "disabled"
        else:
            self.add["state"] = "disabled"
            self.item[1].configure(state="disabled")
            self.category[1].configure(state="disabled")
            self.update["state"] = "normal"
            self.cancel["state"] = "normal"

    def item_changed(self, *args):
        print(self)
        self.price.config(text=0)
        self.description.config(text='')
        self.spice_lvl.config(text=1)
        self.veg.config(text='Yes')
        self.gst.config(text=18)

        helper = ComponentHelper()
        helper.change_text(self.qty, 1)

        for data in self.item_list:
            if data.item == args[0]:
                self.price.config(text=data.price)
                self.description.config(text=data.description)
                if(data.spice_level == 0):
                    self.spice_lvl.config(text='Low')
                elif(data.spice_level == 1):
                    self.spice_lvl.config(text='Medium')
                elif(data.spice_level == 2):
```

```
                    self.spice_lvl.config(text='High')

                if(data.veg == 1):
                    self.veg.config(text='Yes')
                else:
                    self.veg.config(text='No')

                #self.category[0].set(data.category)

    def table_changed(self, *args):
        print(self)
        #self.output_label['text'] = f'You selected: {self.option_var.get()}'

    def order_create(self):
        if self.table[0].get() == "Please Select":
            messagebox.showerror('Failure!', 'Please Select Table Number!')
            return
        if not self.order_details:
            messagebox.showerror('Failure!', 'Please Add atleast one item!')
            return

        db = Database()

        order_header = OrderHeader()
        for data in self.table_list:
            if data.table_number == self.table[0].get():
                order_header.table_id = data.table_id

        order_header.total_amount = self.total_amount.cget("text")

        if(self.order_header_id != '' and self.order_header_id !=
0):
            order_header.order_header_id = self.order_header_id
            db.order_header_update(order_header)
            db.order_detail_delete_by_id(self.order_header_id)
        else:
            self.order_header_id = db.order_header_create(order_header)

        for ord in self.order_details:
            order_detail = OrderDetail()
            order_detail.order_detail_id = ord.order_detail_id
            order_detail.order_header_id = self.order_header_id
            order_detail.item_id = ord.item_id
            order_detail.qty = ord.qty
            order_detail.price = ord.price
            order_detail.amount = ord.amount
            db.order_detail_create(order_detail)
```

```
        messagebox.showinfo('Success!', 'Order Created Successfully')
        self.ViewUpdated()

    def AddItem(self, tv, update):
        if self.table[0].get() == "Please Select":
            messagebox.showerror('Failure!', 'Please Select Table!')
            return

        if self.item[0].get() == "Please Select":
            messagebox.showerror('Failure!', 'Please Select Item Name!')
            return

        if self.qty.get() == "0" or self.qty.get() == "":
            messagebox.showerror('Failure!', 'Quantity Should be greater than
Zero')
            return

        if int(self.qty.get()) > 9:
            messagebox.showerror('Failure!', 'Quantity Should be less than
TEN')
            return

        if int(self.qty.get()) < 0:
            messagebox.showerror('Failure!', 'Quantity Should be greater than
0')
            return

        if(update == False):
            for row in self.order_details:
                if(row.item == self.item[0].get()):
                    messagebox.showerror('Failure!', 'Item already present in
order, to make changes, edit the item')
                    return


        order_detail = OrderDetail()
        for data in self.item_list:
            if data.item == self.item[0].get():
                order_detail.item_id = data.item_id
                order_detail.item = self.item[0].get()

        order_detail.qty = self.qty.get()
        order_detail.price = self.price.cget("text")
        order_detail.gst = self.gst.cget("text")
        order_detail.amount = float(order_detail.qty) *
float(order_detail.price) * (1 + (float(order_detail.gst)/100))

        if(update == True):
```

```
            x = tv.get_children()
            print(x)
            for row in x:
                values = tv.item(row)['values']
                if(values[0] == order_detail.item):
                    tv.item(row, text=tv.item(row)['text'],
values=(order_detail.item, order_detail.qty, order_detail.price,
order_detail.gst, order_detail.amount))
            for row in self.order_details:
                if(order_detail.item == row.item):
                    row.qty = order_detail.qty
                    row.amount = order_detail.amount
        else:
            self.order_details.append(order_detail)
            tv.insert("", 'end', iid=None, text=self.i,
values=(order_detail.item, order_detail.qty, order_detail.price,
order_detail.gst, order_detail.amount))
            self.i = self.i + 1

        total_amount = 0
        for row in self.order_details:
            total_amount = total_amount + row.amount

        self.total_amount.config(text=total_amount)

        self.enable_insert(True)

    def edit(self):
        print("Edit", self.popup.selection)
        self.item[0].set(self.popup.selection['Item'])
        helper = ComponentHelper()
        helper.change_text(self.qty, self.popup.selection['Qty'])
        self.price.config(text=str(self.popup.selection['Price']))

        for data in self.item_list:
            if data.item == self.item[0].get():
                self.category[0].set(data.category)

        self.enable_insert(False)

    def delete(self):
        print("Delete", self.popup.selection)
        try:
            selected_item = self.tv.selection()[0]
            self.tv.delete(selected_item)
        except:
            messagebox.showerror('Failure!', 'Please select an Item before
Deleting!')
```

```
            return

        ord_det = OrderDetail()
        for row in self.order_details:
            if(row.item == self.popup.selection['Item']):
                ord_det = row

        if(ord_det.amount > 0):
            self.order_details.remove(ord_det)

        total_amount = 0
        for row in self.order_details:
            total_amount = total_amount + row.amount

        self.total_amount.config(text=total_amount)

    def do_popup(self, event):
        # display the popup menu
        try:
            self.popup.selection =
self.treeview.set(self.treeview.identify_row(event.y))
            self.popup.post(event.x_root, event.y_root)
        finally:
            # make sure to release the grab (Tk 8.0a1 only)
            self.popup.grab_release()

    def CreateUI(self, win, row):
        #Create menu
        self.popup = tk.Menu(win, tearoff=0)
        self.popup.add_command(label="Edit", command=self.edit)
        self.popup.add_separator()
        self.popup.add_command(label="Delete", command=self.delete)

        self.tv = Treeview(win, height=9)
        self.tv.bind("<Button-3>", self.do_popup)

        self.tv['columns'] = ('Item', 'Qty', 'Price', 'GST', 'Amount')

        self.tv.heading("#0", text='S No', anchor='w')
        self.tv.column("#0", anchor="w", width=25)

        self.tv.heading('Item', text='Item')
        self.tv.column('Item', anchor='center', width=200)

        self.tv.heading('Qty', text='Qty')
        self.tv.column('Qty', anchor='center', width=100)

        self.tv.heading('Price', text='Price')
```

```
        self.tv.column('Price', anchor='center', width=100)

        self.tv.heading('GST', text='GST')
        self.tv.column('GST', anchor='center', width=100)

        self.tv.heading('Amount', text='Amount')
        self.tv.column('Amount', anchor='center', width=100)

        self.tv.grid(row=row, column=0, sticky = (N,S,W,E), columnspan=2)
        self.treeview = self.tv

        return self.tv
```

table_create.py

```
import tkinter as tk

from model.table import Table
from view.helper.comp_helper import ComponentHelper

class TableCreate():
    def __init__(self, win, tbl=Table):
        print('create customer constructor')
        self.createWidgets(win, tbl)

    def createWidgets(self, win, tbl=Table):
        top=win.winfo_toplevel()

        top.rowconfigure(0, weight=1)
        top.columnconfigure(0, weight=1)

        win.rowconfigure(1, weight=1)
        win.columnconfigure(1, weight=1)

        helper = ComponentHelper()
        helper.create_label_entry(win, 0,'Table Number', tbl.table_number)
        helper.create_label_entry(win, 1,'Number of Seats',
tbl.number_of_seats)
        helper.create_label_entry(win, 2,'Location', tbl.location)

        self.submit = tk.Button(win, text='Submit', command=win.quit)
        self.submit.grid(row=6, column=1, sticky=tk.N+tk.S+tk.E+tk.W)
```

table_reservation_create.py

```
import tkinter as tk
from tkinter import *

from datetime import datetime, timedelta
from tkinter import messagebox
from tkinter.ttk import Treeview
from model.table_reservation import TableReservation
from view.helper.comp_helper import ComponentHelper
from database.db import Database
import sys
from view.event import Event


class TableReservationCreate():
    customer = tk.OptionMenu
    table = tk.OptionMenu
    datetime = ''
    pax = 2
    popup = tk.Menu
    tv = Treeview
    id = 0

    def __init__(self, win):
        print('create table reservation constructor')
        self.OnViewUpdated = Event()

    def ViewUpdated(self):
        self.OnViewUpdated()

    def AddSubscribersForViewUpdatedEvent(self,objMethod):
        self.OnViewUpdated += objMethod

    def RemoveSubscribersForViewUpdatedEvent(self,objMethod):
        self.OnViewUpdated -= objMethod

    def createWidgets(self, win, id=0):
        top=win.winfo_toplevel()

        top.rowconfigure(0, weight=1)
        top.columnconfigure(0, weight=1)

        win.rowconfigure(1, weight=1)
        win.columnconfigure(1, weight=1)

        helper = ComponentHelper()
        win.frame = helper.add_background(win,
"./images/table_resev_add_del.gif", 0.90)

        db = Database()
```

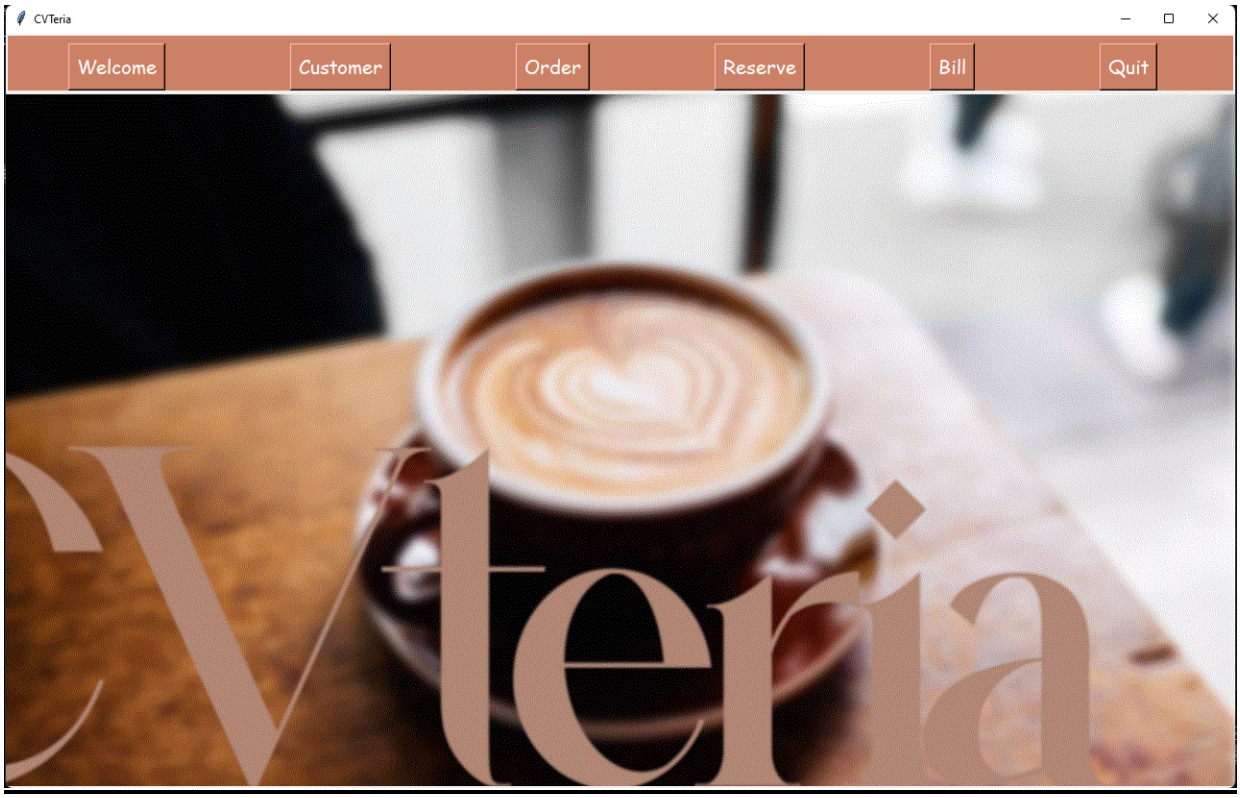
```
        self.table_list = db.table_get_all()
        table_arr = []
        table_arr.append('Please Select')
        for data in self.table_list:
            table_arr.append(data.table_number)

        self.cust_list = db.customer_get_all()
        cust_arr = []
        cust_arr.append('Please Select')
        for data in self.cust_list:
            cust_arr.append(data.customer_name)

        obj = TableReservation()

        db = Database()
        if(id > 0):
            self.id = id
            list = db.table_reservation_get_by_id(id)
            for data in list:
                obj.reservation_id=data.reservation_id
                obj.table_id = data.table_id
                obj.table_number = data.table_number
                obj.pax = data.pax
                obj.datetime = data.datetime
                obj.customer_id = data.customer_id
                obj.customer = data.customer

        for data in self.table_list:
            if(data.table_number == obj.table_number):
                obj.location = data.location
                obj.number_of_seats = data.number_of_seats

        helper = ComponentHelper()
        self.customer = helper.create_label_options_menu(win.frame,
0,'Customer', cust_arr, self.item_changed, obj.customer)
        self.table = helper.create_label_options_menu(win.frame, 1,'Table',
table_arr, self.table_changed, obj.table_number)
        self.number_of_seats = helper.create_label_label(win.frame, 2,'Number
of Seats', obj.number_of_seats)
        self.location = helper.create_label_label(win.frame, 3,'Location',
obj.location)
        self.datetime = helper.create_label_entry(win.frame, 4,'DateTime ',
obj.datetime)
        self.pax = helper.create_label_entry(win.frame, 5,'Pax', obj.pax)

        self.submit = tk.Button(win.frame, text='Cancel',
command=lambda:self.table_reservation_cancel())
        self.submit.grid(row=6, column=0, sticky=tk.N+tk.S+tk.E+tk.W)
```

```
        self.submit = tk.Button(win.frame, text='Submit',
command=lambda:self.table_reservation_create())
        self.submit.grid(row=6, column=1, sticky=tk.N+tk.S+tk.E+tk.W)

    def table_changed(self, *args):
        for data in self.table_list:
            if(data.table_number == args[0]):
                self.number_of_seats.config(text=data.number_of_seats)
                self.location.config(text=data.location)

    def table_reservation_cancel(self):
        self.ViewUpdated()

    def table_reservation_create(self):
        if self.customer[0].get() == "Please Select":
            messagebox.showerror('Failure!', 'Please Select Customer!')
            return
        if self.table[0].get() == "Please Select":
            messagebox.showerror('Failure!', 'Please Select Table!')
            return
        if self.datetime.get().strip() == "":
            messagebox.showerror('Failure!', 'Please Enter DateTime!')
            return

        try:
            res = bool(datetime.strptime(self.datetime.get().strip(), '%Y-%m-
%d  %H:%M:%S'))
        except BaseException as e:
            print(e)
            messagebox.showerror('Failure!', 'Please enter DateTime in YYYY-
mm-dd HH:MM:SS format')
            return

        if datetime.strptime(self.datetime.get().strip(), '%Y-%m-%d %H:%M:%S')
< datetime.now():
            messagebox.showerror('Failure!', 'DateTime should be in the
future')
            return
        if self.pax.get() == "":
            messagebox.showerror('Failure!', 'Please Enter the number of
people!')
            return
        if self.pax.get() == "0":
            messagebox.showerror('Failure!', 'Please Enter at least 1 person')
            return
        if int(self.pax.get()) > int(self.number_of_seats.cget("text")):
```

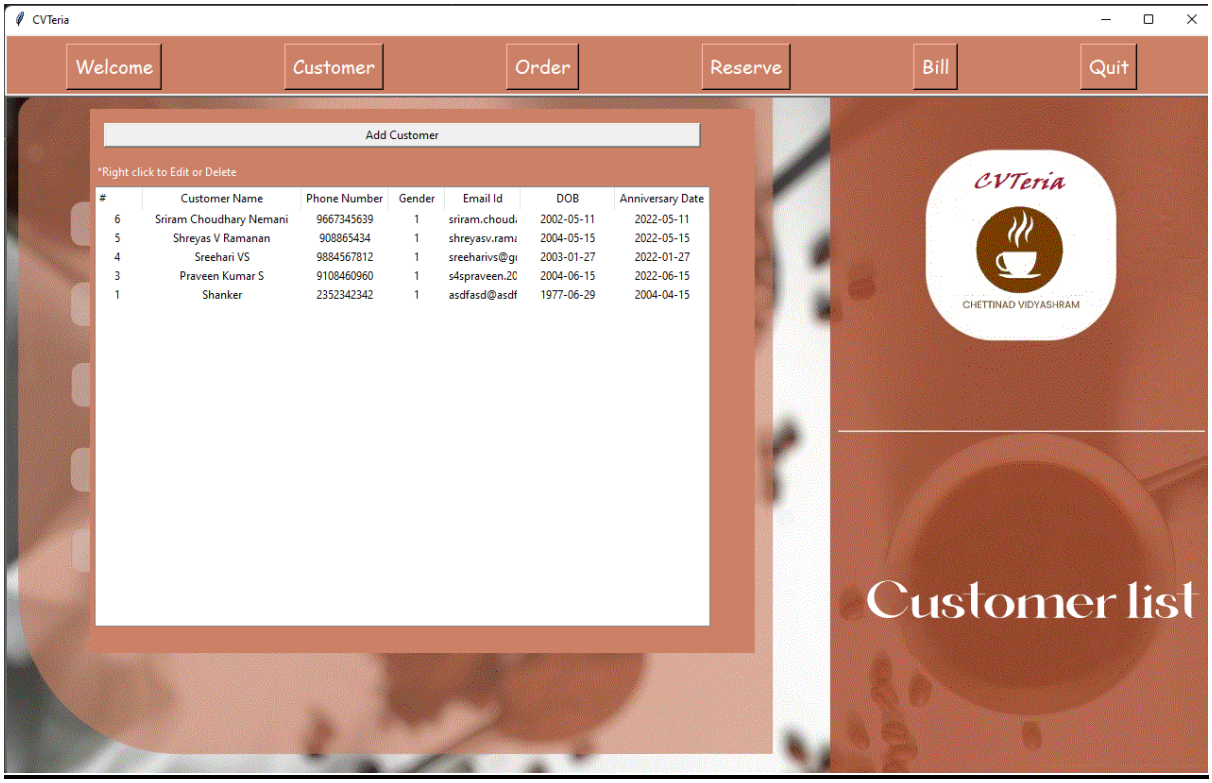
```
            messagebox.showerror('Failure!', str(self.table[0].get()) + ' can
accomodate only '+ str(self.number_of_seats.cget("text")) + ' people')
            return


        db = Database()
        if(self.id == 0):
            reserv_list = db.table_reservation_get_all()
            for data in reserv_list:

                dt = datetime.strptime(str(data.datetime), '%Y-%m-%d
%H:%M:%S')
                dt_one = dt + timedelta(hours=1)
                user_dt = datetime.strptime(self.datetime.get().strip(), '%Y-
%m-%d %H:%M:%S')

                if((dt <= user_dt <= dt_one) and data.table_number ==
self.table[0].get()):
                    messagebox.showerror('Failure!', 'Table already booked
from ' + str(dt) + " to " + str(dt_one))
                    return

        reserv = TableReservation()
        for data in self.table_list:
            if data.table_number == self.table[0].get():
                reserv.table_id = data.table_id

        for data in self.cust_list:
            if data.customer_name == self.customer[0].get():
                reserv.customer_id = data.customer_id

        reserv.pax = self.pax.get()
        reserv.datetime = self.datetime.get()

        try:
            if(self.id > 0):
                reserv.reservation_id = self.id
                db.table_reservation_update(reserv)
            else:
                db.table_reservation_create(reserv)
            messagebox.showinfo('Success!', 'Table Reserved Successfully')
        except:
            type, value, traceback = sys.exc_info()
            messagebox.showerror('Failure!', value)

        self.ViewUpdated()

    def item_changed(self, *args):
        print(self)
```

Sample Output:

Welcome Screen:



Customer Listing Screen:



Order Listing Screen:

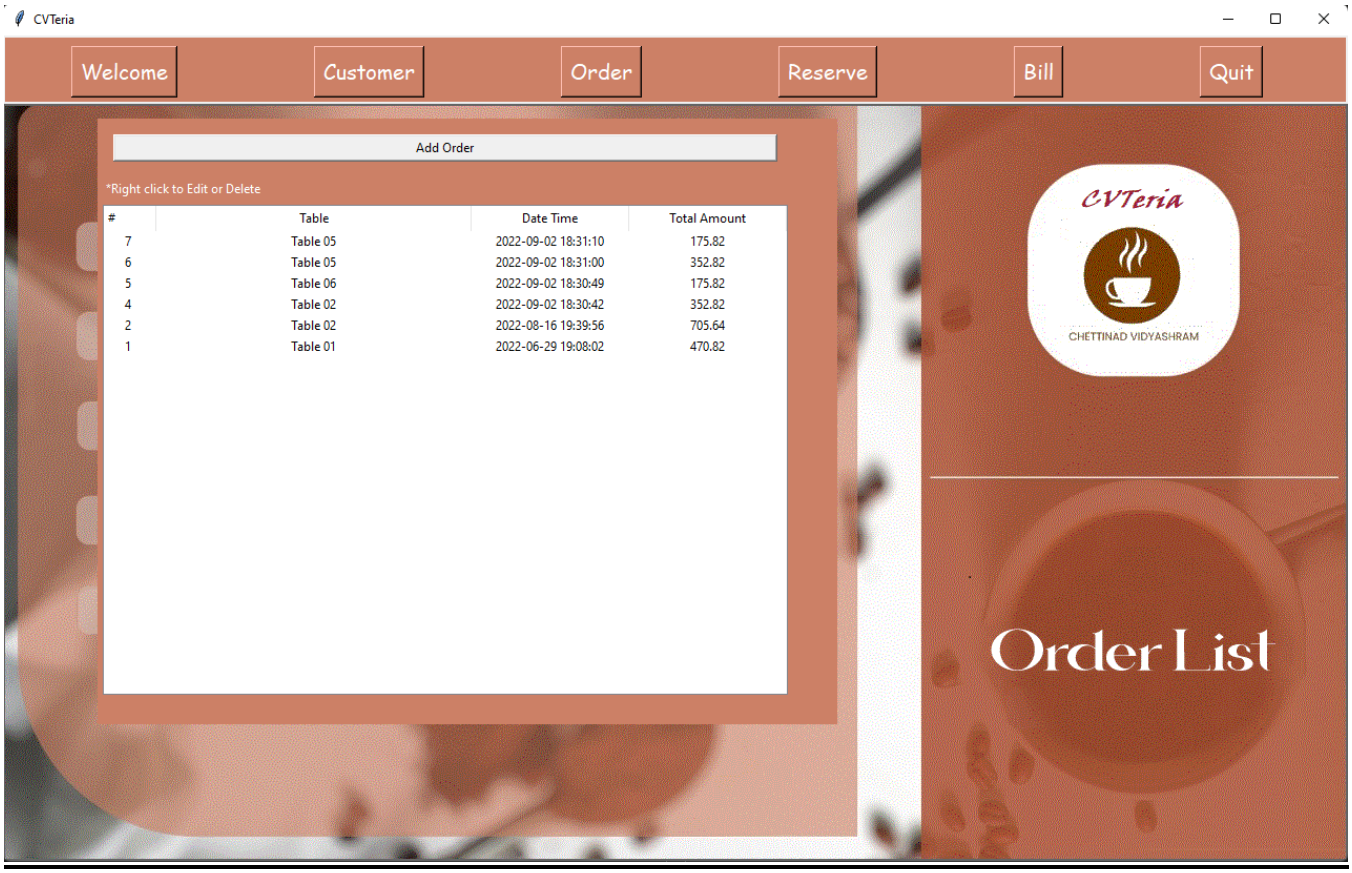
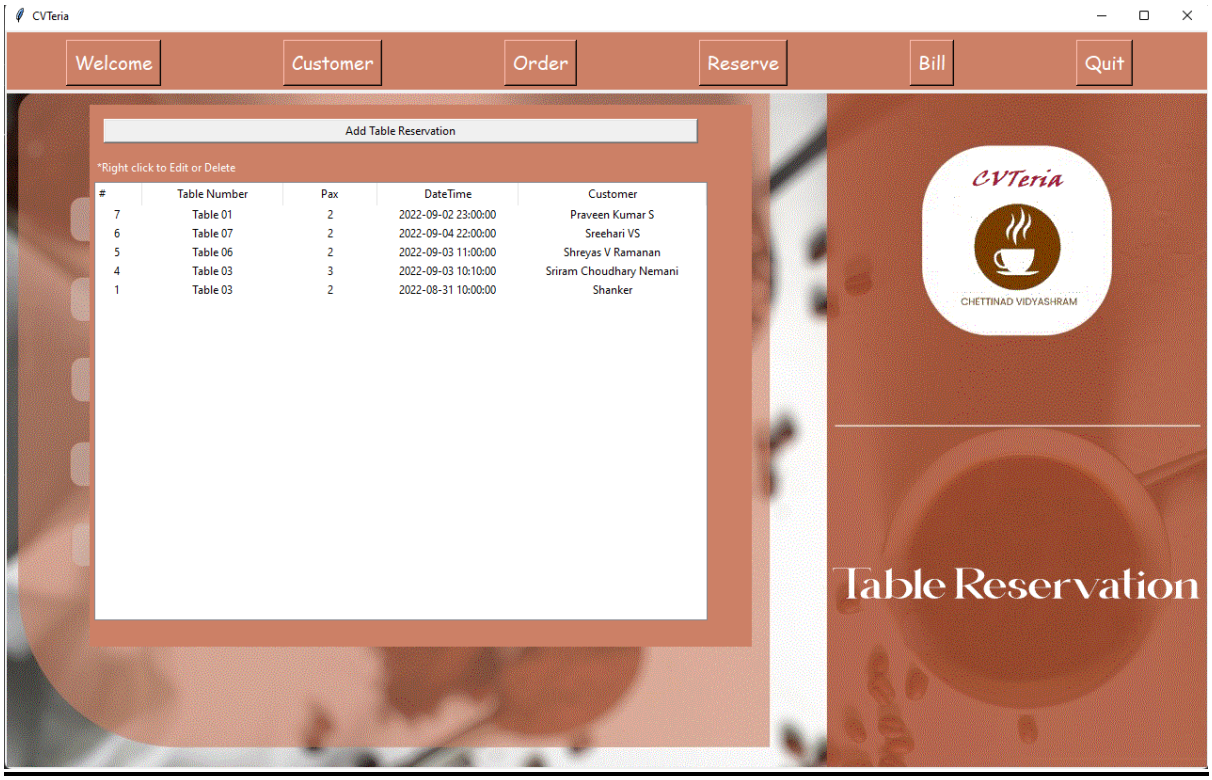
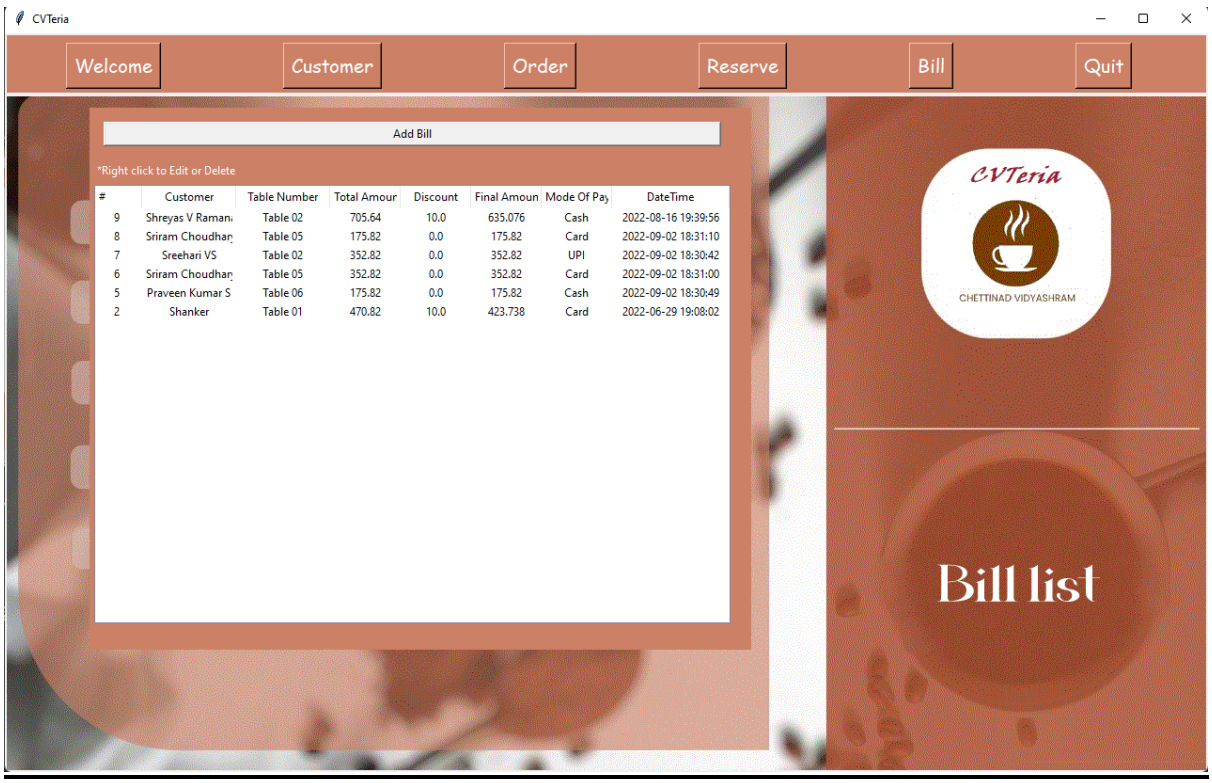


Table Reservation Screen:

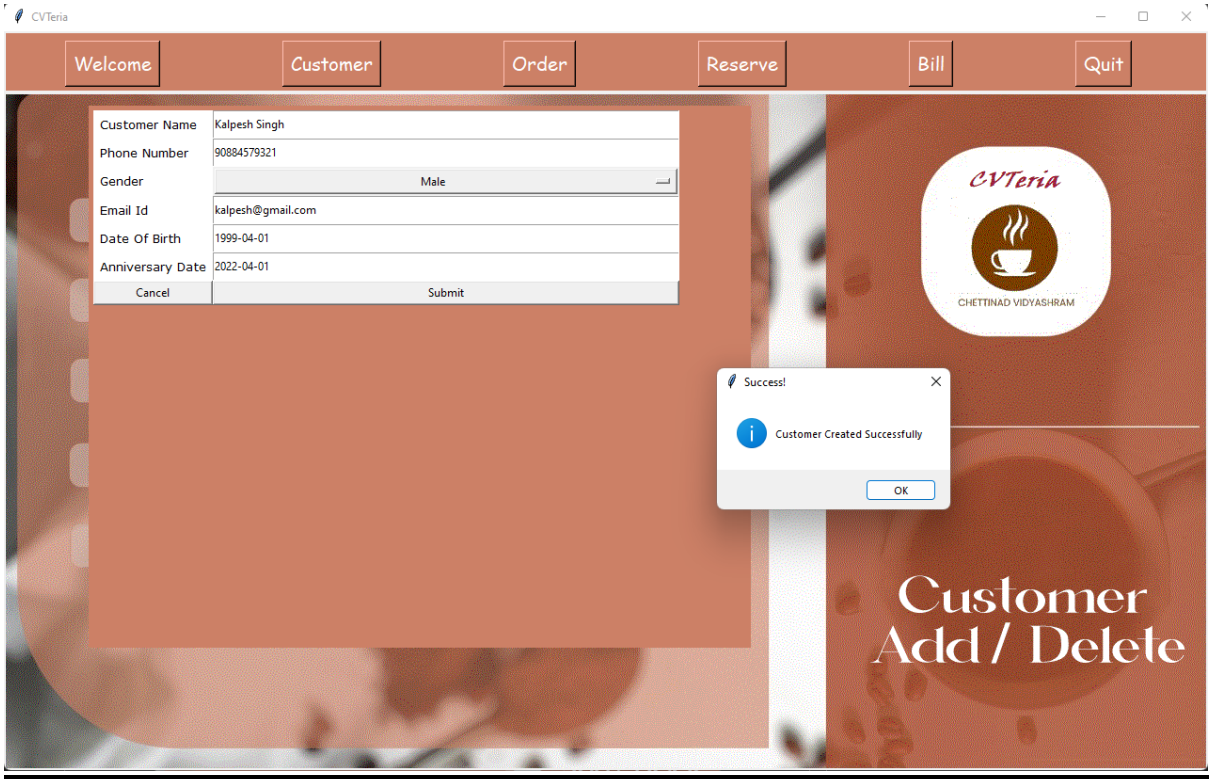


Bill Listing Screen:

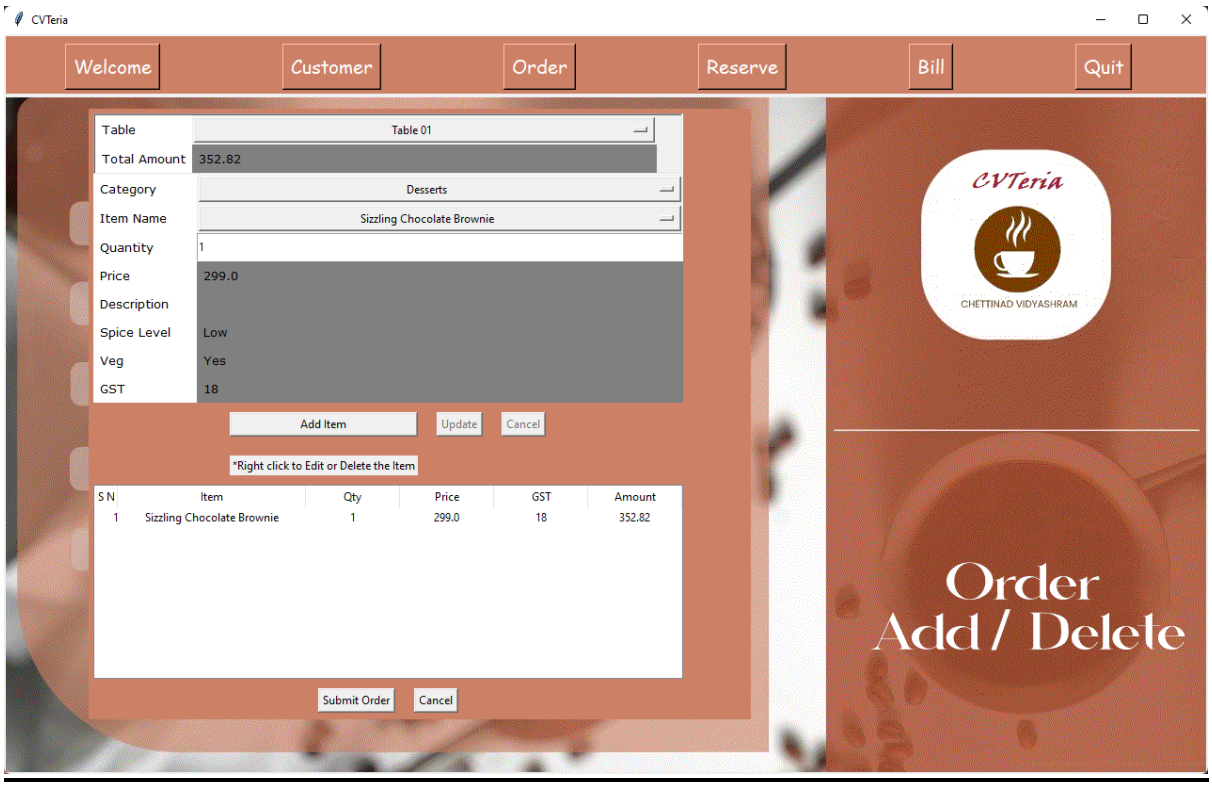


Create Customer Screen:

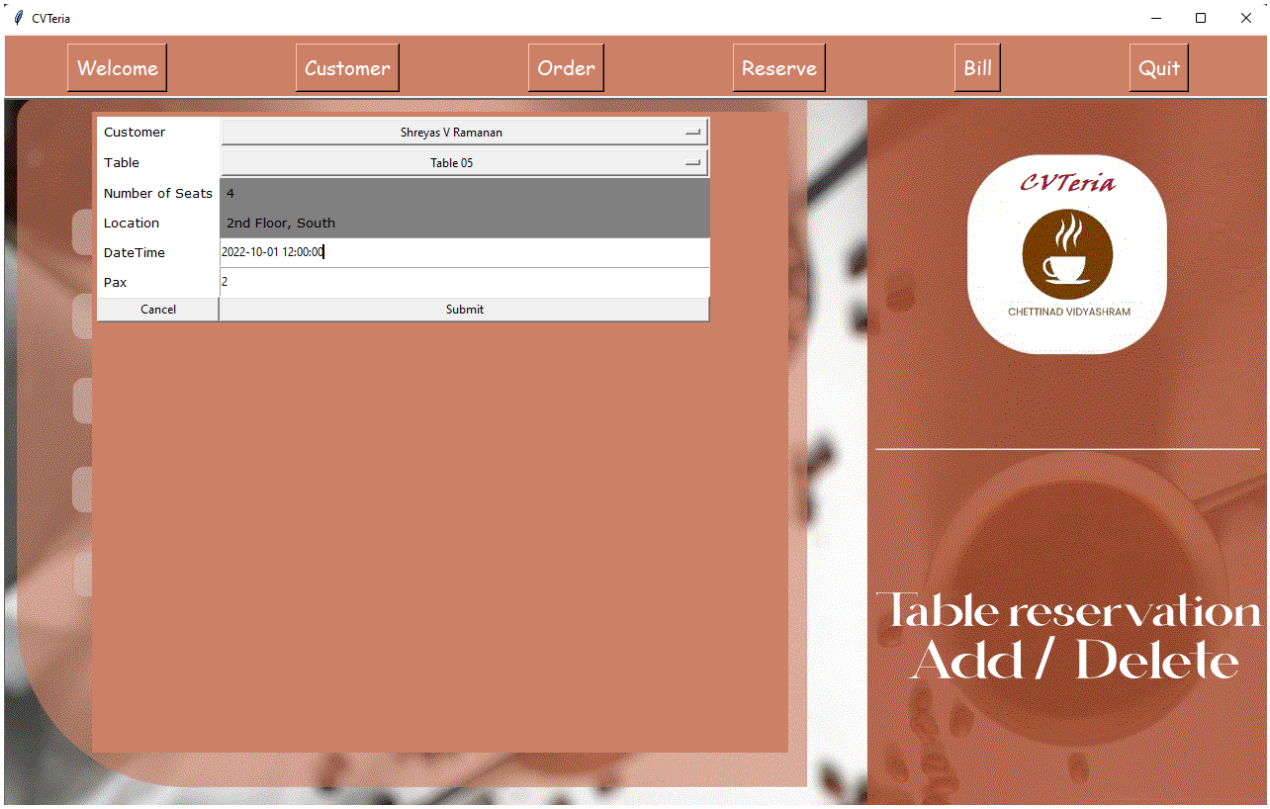
“Customer Created Successfully” message is displayed



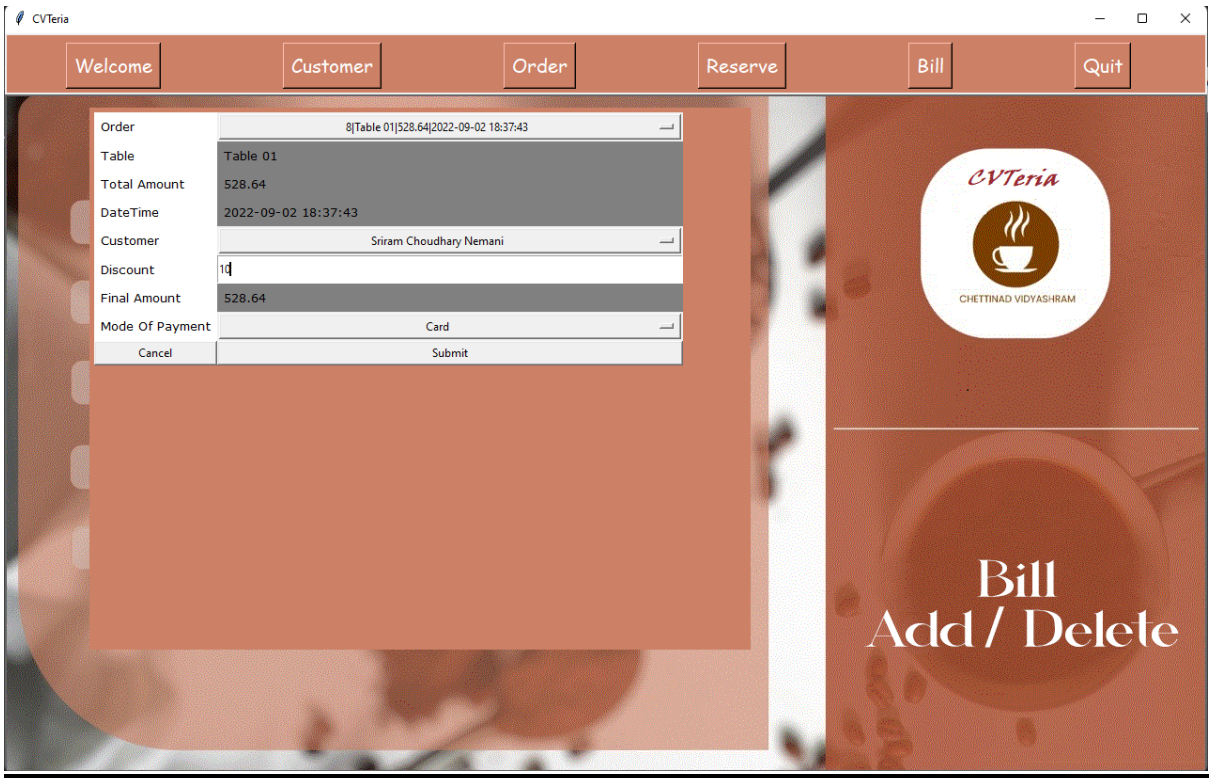
Create Order Screen:



Create Table Reservation Screen:

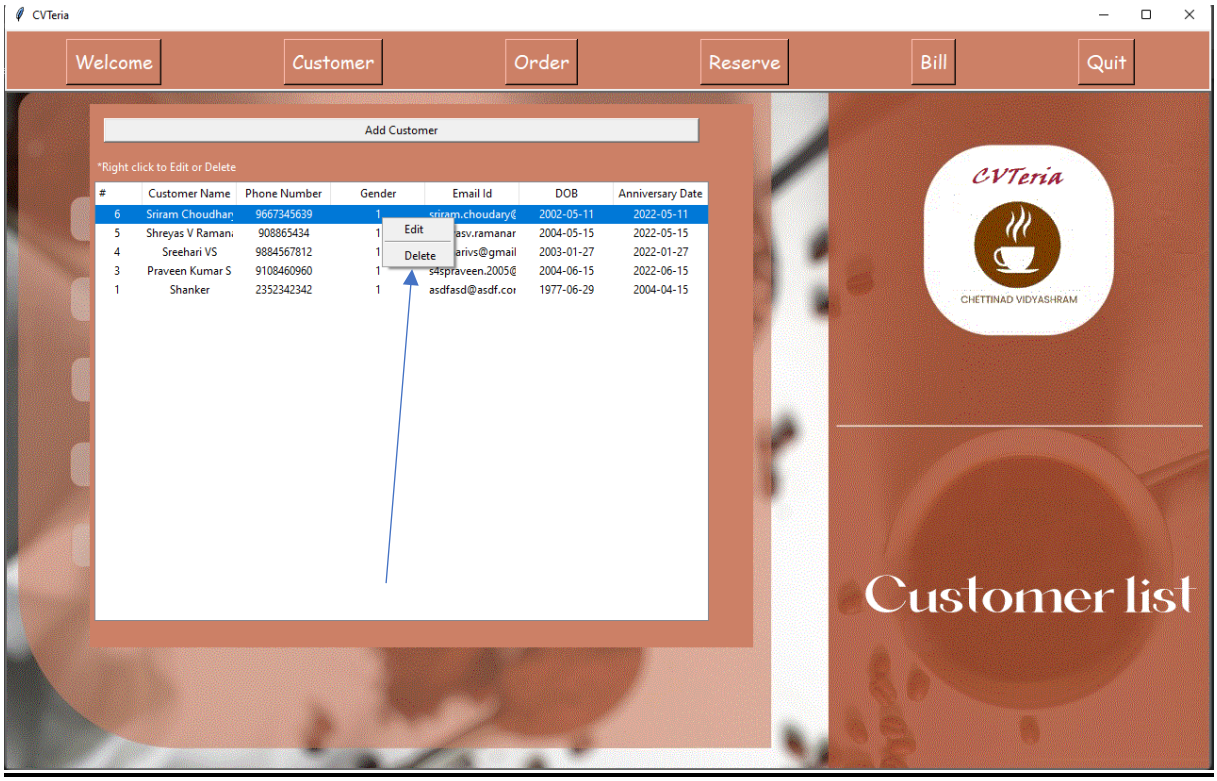


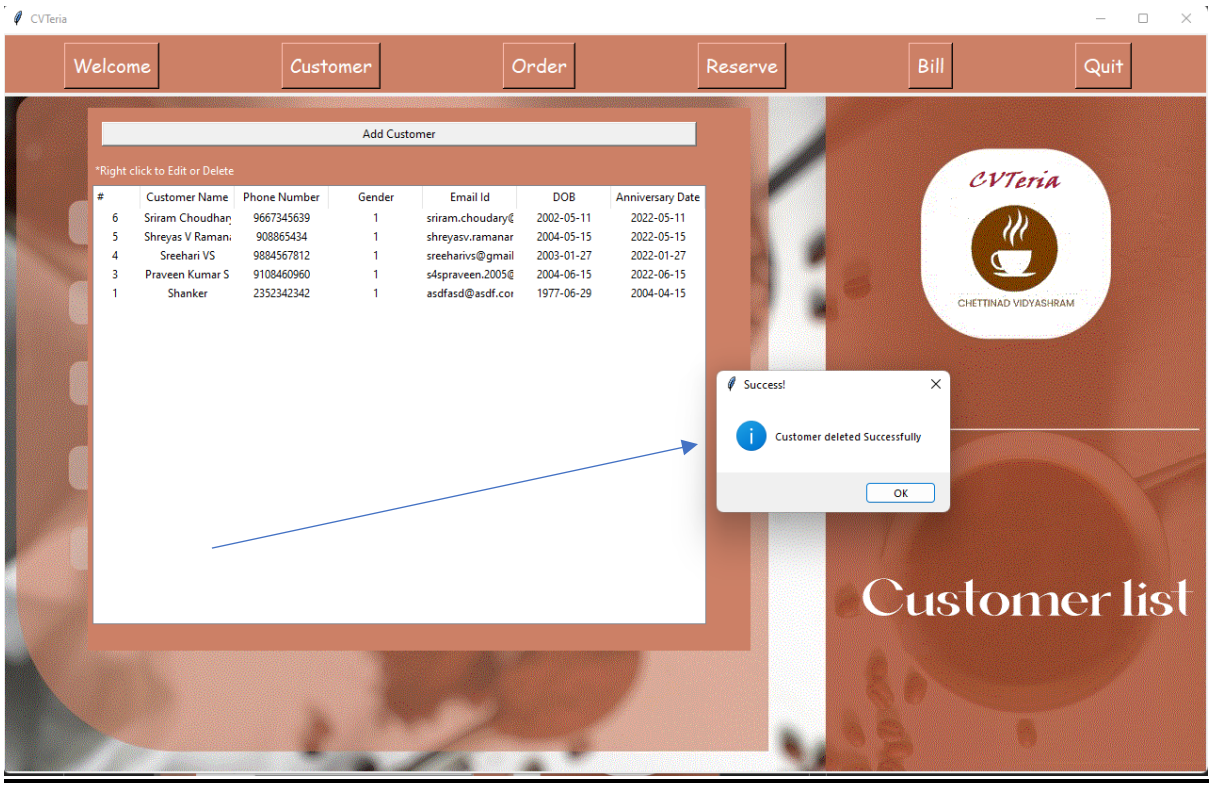
Create Bill Screen:



Edit/Delete Customer (Enabled on right clicking):

Customer Deleted successfully message is displayed

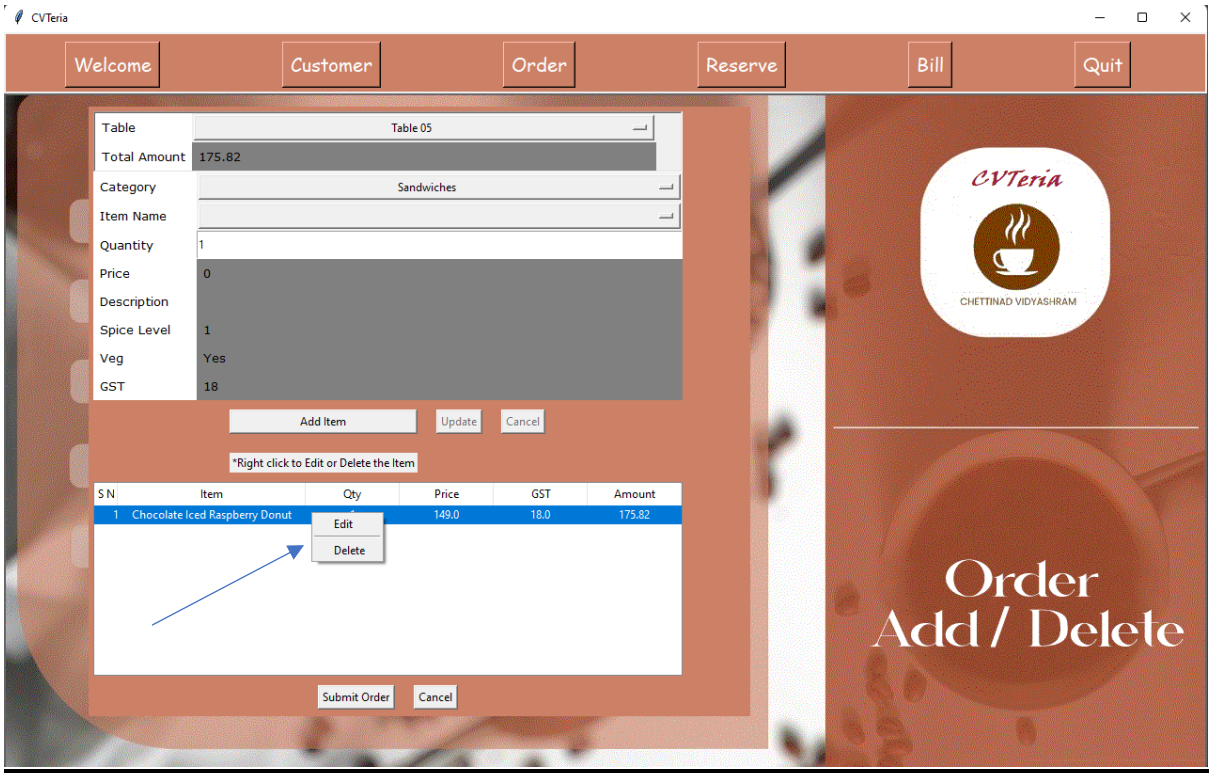




Note: Editing and Deleting and enable for all screens.

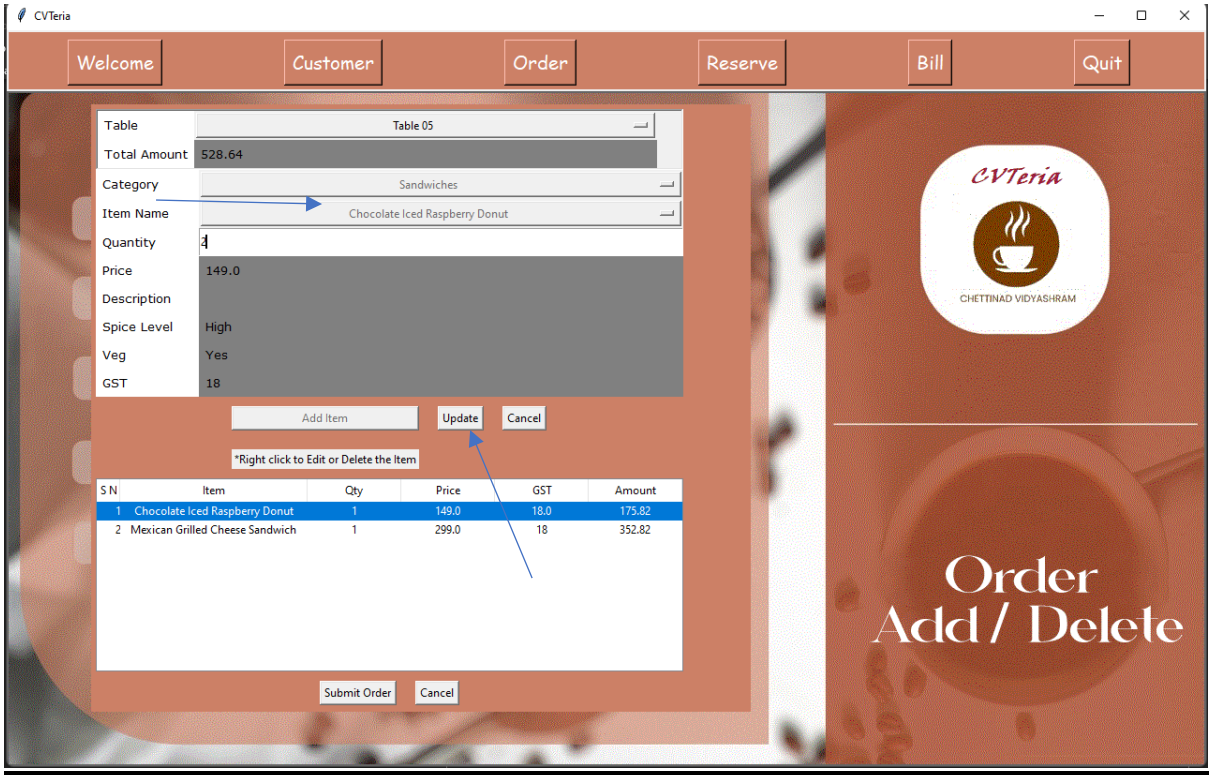
Message boxes are also enabled for every screen.

Edit Specific Item in an order:



On Editing a specific item, the program only allows you to change the quantity and nothing else

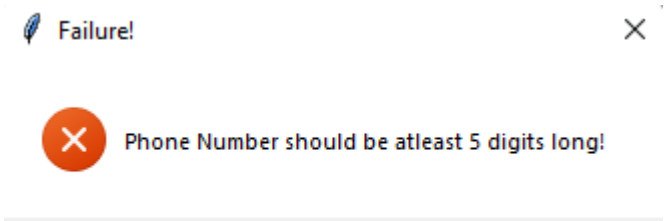
The update button becomes active since we are only editing and not adding a new item…

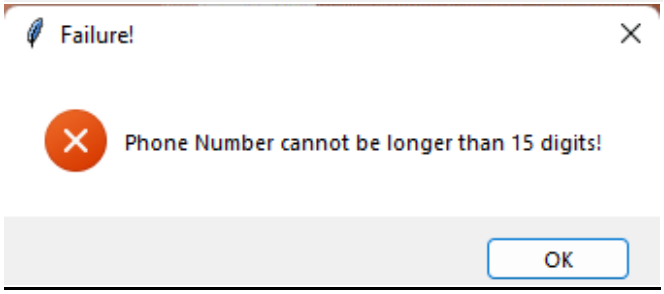


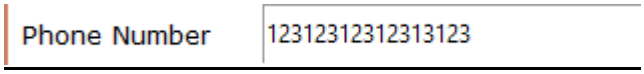
Validation:

The following validations have been enabled in Customer screen:

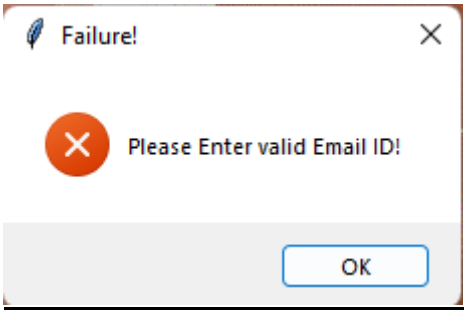
1. Phone Number: Phone number should be at least 5 digits long and it can't be more than 15 digits long.

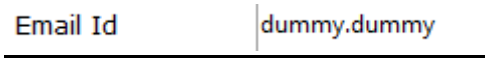




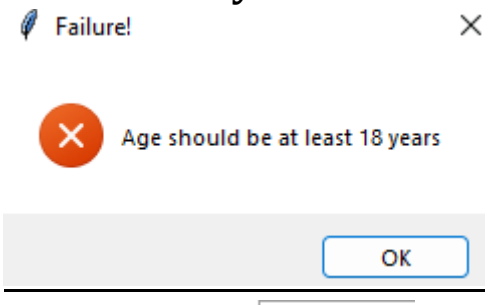


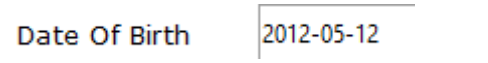
2. Email ID: Email ID should be of the form abc@abc.com

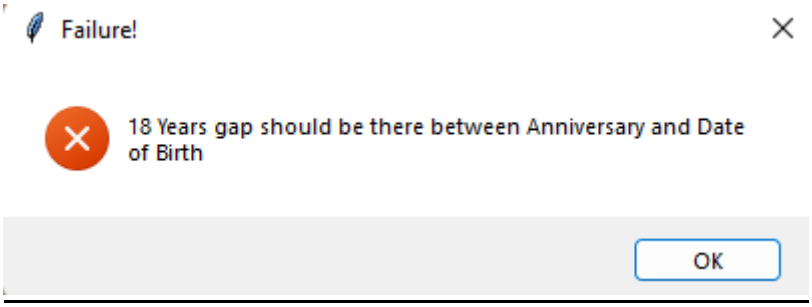


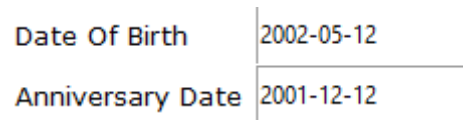


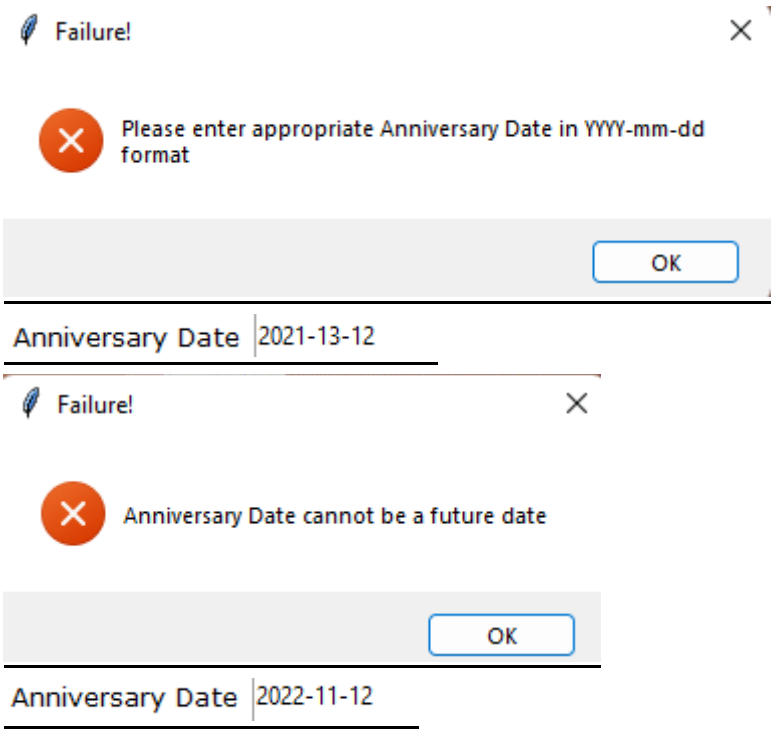
3. Date Validation: Minimum Age to create an account in our cafeteria is 18 years.
   Difference between birth date and anniversary should be at least 18 years.
   Date should be valid in YYYY-mm-dd format.
   Anniversary date cannot be a future date.





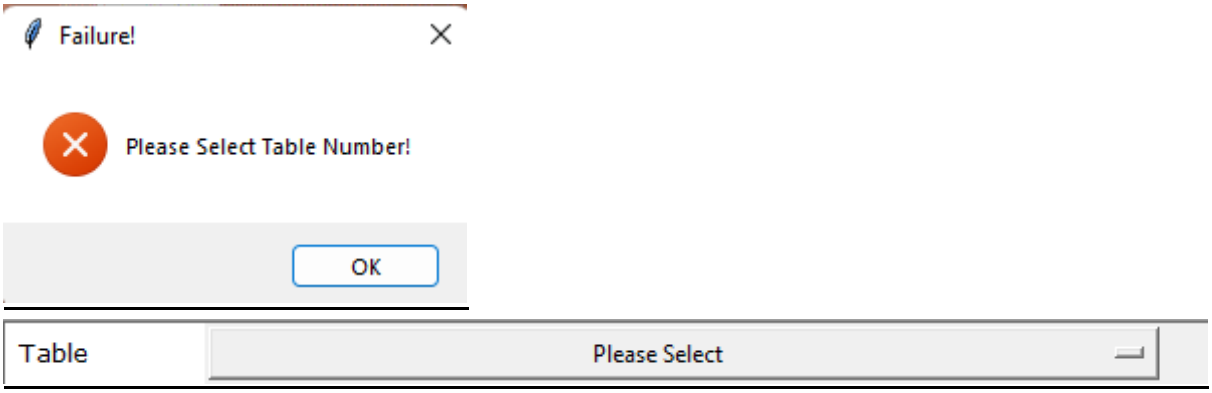




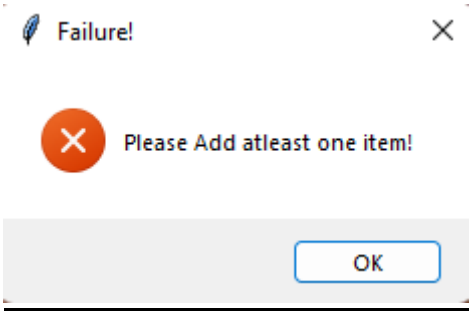


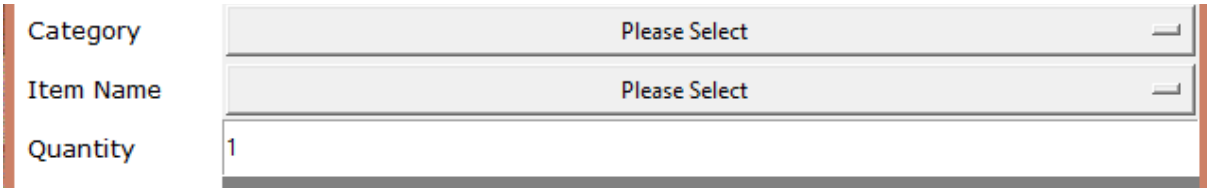
The following validations have been added in order screen:

1. Table Number: One cannot submit order without selecting table number.



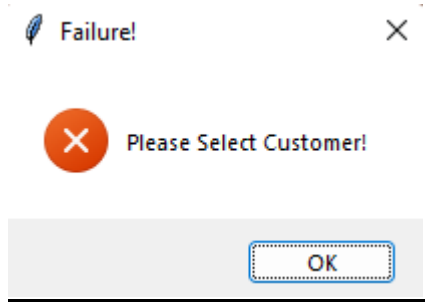
2. Items: The minimum number of items to be added for an order to be placed is 1.



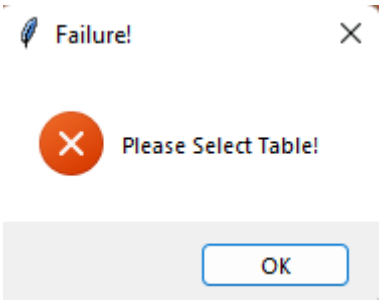


The following validations have been added to table reservation screen:

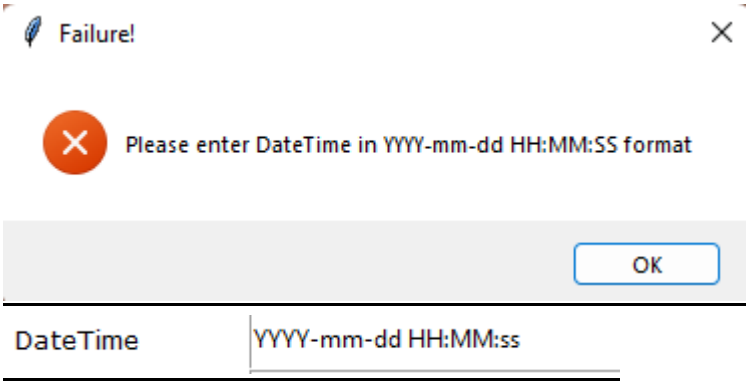
1. Customer: One cannot reserve a table without selecting which customer.



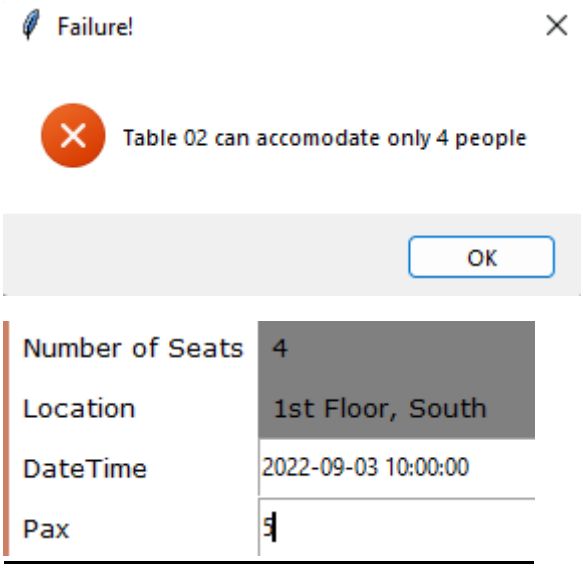
2. Table: One cannot reserve a table without selecting the table.



3. Date Time: Date and time should be entered in valid format. Date should be a future date and time.

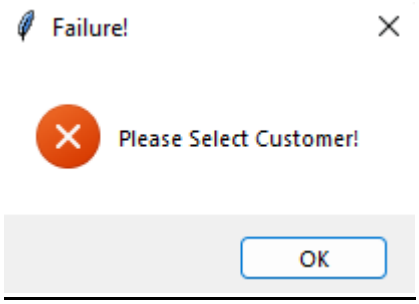


4. Number of People: The number of people the reservation is being made for should not exceed the number of seats at the table.



The following validations have been added to Bill screen:

1. Customer: To Create a bill, customer should be selected.


Failure!
Please Select Customer!
OK

CONCLUSION

Scope Of Improvement:

The application can be improved by integrating with other Cafeteria Management applications and also further improvement can be made on the GUI with additional features. An online interface can be made available so that it can be accessed from PC and Mobile. The security of the system can further be improved by using other authorization solutions.

BIBLIOGRAPHY

1. Stackoverflow.com
2. W3schools.com
3. Javatpoint.com
4. Youtube.com