CAPSTONE PROJECT REPORT

(Project Term January-May 2023)

RESTAURANT MANAGEMENT SYSTEM

Submitted by

Jaswanth Singh Kumar Lankadasu **Registration Number: 12104841**

Project Group Number: K21UPA08

Course Code: INT216

Under the Guidance of

Waseem Ud Din Wani

School of Computer Science and Engineering

DECLARATION

We hereby declare that the project work entitled Restaurant Management System is an authentic record of my own work carried out as requirements of Capstone Project for the award of B. Tech degree in School of Computer Science and Engineering from Lovely Professional University, Phagwara, under the guidance of Waseem Ud Din Wani, during April to May 2023. All the information furnished in this capstone project report is based on my own intensive work and is genuine.

Project Group Number: K21UPA08

Name of Student: Jaswanth Singh Kumar Lankadasu
Registration Number: 12104841

Jaswanth Singh Kumar Lankadasu
Date: 03/05/2023

CERTIFICATE

This is to certify that the declaration statement made by the student is correct to the best of my knowledge and belief. He have completed this Capstone Project under my guidance and supervision. The present work is the result of their original investigation, effort, and study. No part of the work has ever been submitted for any other degree at any University. The Capstone Project is fit for the submission and partial fulfillment of the conditions for the award of B. Tech degree in School of Computer Science and Engineering from Lovely Professional University, Phagwara.

Signature and Name of the Mentor

Designation

School of Computer Science and Engineering,
Lovely Professional University,
Phagwara, Punjab.

Date:

ACKNOWLEDGEMENT

I would like to thank my project supervisor **Waseem Ud Din Wani** for providing me with guidance and support throughout the project. Their valuable suggestions and feedback were instrumental in shaping the project into its current form.

I would also like to extend my gratitude to my friends and family for their unwavering support and encouragement throughout this project. Their constant motivation kept me going during challenging times.

I am grateful to Yaswanth and Tanu for their help and support during the project's development. Their insights and suggestions were valuable in making this project a success.

Finally, I would like to express my sincere thanks to all the resources from the various books, research papers, and online sources, which proved to be instrumental in the completion of this project.

Thank you all for your support and contribution to this project.

TABLE OF CONTENTS

1. INTRODUCTION

A Restaurant Management System is a software application that helps restaurant owners manage their business operations more efficiently. The system provides an easy-to-use interface for managing menu items, order information, among other things.
Restaurant Management System has the potential to greatly improve the efficiency and profitability of restaurants. By automating many routine tasks, your software can help restaurant owners to save time and reduce costs. Moreover, the system can provide valuable insights into customer behaviour, helping restaurant owners to optimize their menus, prices, and marketing strategies.

2. PROFILE OF THE PROBLEM

The restaurant industry is a highly competitive sector, and in order to remain successful, restaurant owners need to focus on improving customer experience and streamlining their operations. However, traditional methods of managing restaurant orders and billing can be inefficient and prone to errors, leading to longer wait times and decreased customer satisfaction. Furthermore, manual record-keeping can be time-consuming and difficult to manage, making it challenging for restaurant owners to track their finances and make informed decisions about their business.

The development of a restaurant management system can address these challenges by automating the ordering and billing process, improving accuracy, and reducing wait times. Additionally, a well-designed restaurant management system can provide restaurant owners with real-time data about their business.

The scope of this study is to design and develop a restaurant management system that meets the specific needs of restaurant owners and their customers. The system should be user-friendly, reliable, and secure, providing a simple and efficient way for customers to place orders and receive bills, and for restaurant.

3. EXISTING SYSTEM

3.1. INTRODUCTION

The existing system for managing restaurant orders and billing is typically a manual process involving handwritten orders, physical menus, and paper receipts. This process is prone to errors and can result in delays, which can negatively impact the customer experience.

3.2. EXISTING SOFTWARE

There are many restaurant management software available in the market, but they may not be suitable for all types of restaurants. For example, some software may be too expensive or too complex for small-scale restaurants. Additionally, many of these software may have features that are not required by a particular restaurant, which can result in unnecessary costs and complexity.

3.3. DFD FOR PRESENT SYSTEM

A data flow diagram (DFD) for the present system can be created to illustrate the flow of data and processes involved in the traditional method of managing orders and billing. The DFD can help identify inefficiencies in the existing system and provide insights into how the new system can be designed.

3.4. WHAT'S NEW IN THE SYSTEM TO BE DEVELOPED

The new restaurant management system will be a software-based solution that automates the process of order taking, food preparation, and billing. The new system will offer several advantages over the existing system, including:

- Improved accuracy: The system will reduce errors in the order taking and billing process, which can lead to more accurate bills and fewer customer complaints.
- Faster service: The system will streamline the ordering and billing process, resulting in faster service and reduced waiting times for customers.
- Better inventory management: The system will enable restaurant owners to track inventory levels in real-time, which can help them optimize their ordering and reduce waste.
- Enhanced customer experience: The system will provide a more modern and convenient ordering experience for customers, which can result in higher customer satisfaction and repeat business.

4. PROBLEM ANALYSIS

4.1. PRODUCT DEFINITION

The product definition for your restaurant management system includes the following features:

- A user-friendly interface that enables customers to place their orders easily.
- A database to store menu items.

- A billing system that calculates the total cost of the order and generates a bill for the customer.
- A reset button that enables the user to clear the order list and start over.
- A close button that allows the user to exit the application.
- Error handling and validation checks to prevent input errors and ensure accuracy..

4.2. FEASIBILITY ANALYSIS

A feasibility analysis is a crucial step in determining the viability of the project. This analysis assesses the technical, operational, and economic feasibility of the project. Technical feasibility evaluates whether the required technology is available and can be implemented effectively. Operational feasibility assesses whether the proposed system can be integrated with existing systems and whether it will be accepted by the users. Economic feasibility determines whether the project is financially viable and whether it can generate a sufficient return on investment.

The feasibility analysis for your restaurant management system project includes the following considerations:

- Technical feasibility: Can the system be developed with the available resources and technology (Tkinter and Python)?
- Operational feasibility: Will the system be easy to use and integrate with existing restaurant processes?
- Economic feasibility: Is the project cost-effective and will the benefits outweigh the costs?

4.3. PROJECT PLAN

The project plan outlines the scope, timeline, resources, and deliverables of the project. This plan includes a detailed breakdown of the project tasks, timelines, dependencies, and milestones. It also identifies the roles and responsibilities of the project team and outlines the communication and risk management plan.

5. SOFTWARE REQUIREMENT ANALYSIS

5.1. INTRODUCITON

The software requirement analysis involves identifying the hardware and software requirements of the system.

5.1.1. Hardware Requirements

Any system with minimum of 4GB RAM and 128GB ROM which can run at least VS Code or Python or any other applications that can able to run the python code freely.

5.1.2. Software Requirements

Any software that is capable of running the python application is fine. Any OS can be fine.

5.2. GENERAL DESCRIPTION

The restaurant management system is designed to provide an easy-to-use interface for restaurant customers and staff. The system will allow customers to view the menu, place orders, and receive bills.

5.3. SPECIFIC REQUIREMENTS

The specific requirements outline the functional and non-functional requirements of the system, such as the user interface, database management, and security features.

5.3.1. User Interface

The user interface of the system should be intuitive and easy to use for both customers and staff. The system should be able to display the menu items in a visually appealing way, with images and descriptions of each item. Customers should be able to add items to their order, view their order details, and receive their bills with ease.

5.3.2. Security

The system should be secure and protect sensitive data such as customer information, order details, and payment information.

5.3.3. Compatibility

The system should be compatible with different hardware and software configurations. The system should be able to run on different operating systems and browsers and should be compatible with different screen sizes and resolutions.

6. DESIGN

6.1. SYSTEM DESIGN

The system design of the restaurant management system can be broken down into several high-level modules, including the user interface, database management, and billing module. The user interface module is responsible for displaying the menu items and enabling customers to place orders. The database management module stores information about menu items, orders, and billing. The billing module calculates the total bill based on the items ordered and generates an invoice for the customer.

6.2. DESIGN NOTATIONS

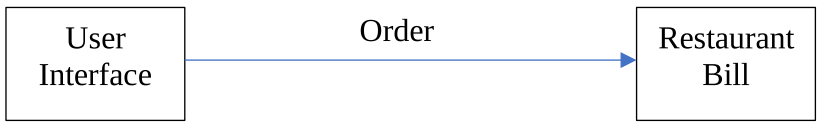
The system design can be illustrated using design notations such as flowcharts, UML diagrams, and pseudocode. A flowchart can be used to show the flow of control between the different modules of the system. A UML diagram can be used to show the classes, objects, and relationships between the different components of the system. Pseudocode can be used to illustrate the algorithms used to implement the system.

6.3. DETAILED DESIGN

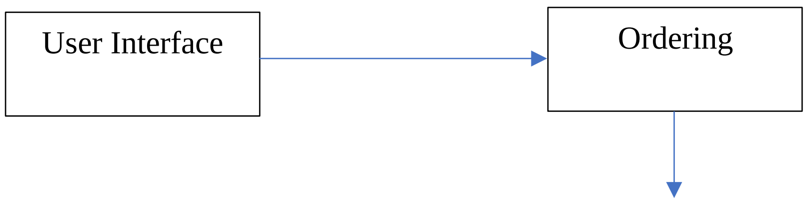
The detailed design of the system involves breaking down the high-level modules into smaller components and defining the algorithms, data structures, and programming languages to be used. For example, the user interface module can be broken down into smaller components such as the menu display, order form, and checkout form. The menu display component can be implemented using the Tkinter GUI toolkit, while the order form can be implemented using Python data structures such as lists and dictionaries. The billing module can be implemented using Python arithmetic operators and functions.

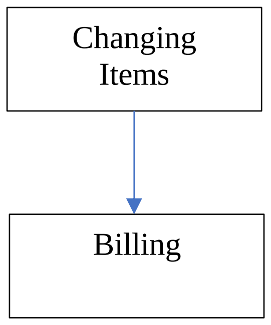
6.3.1. DFD

Level 0 DFD:

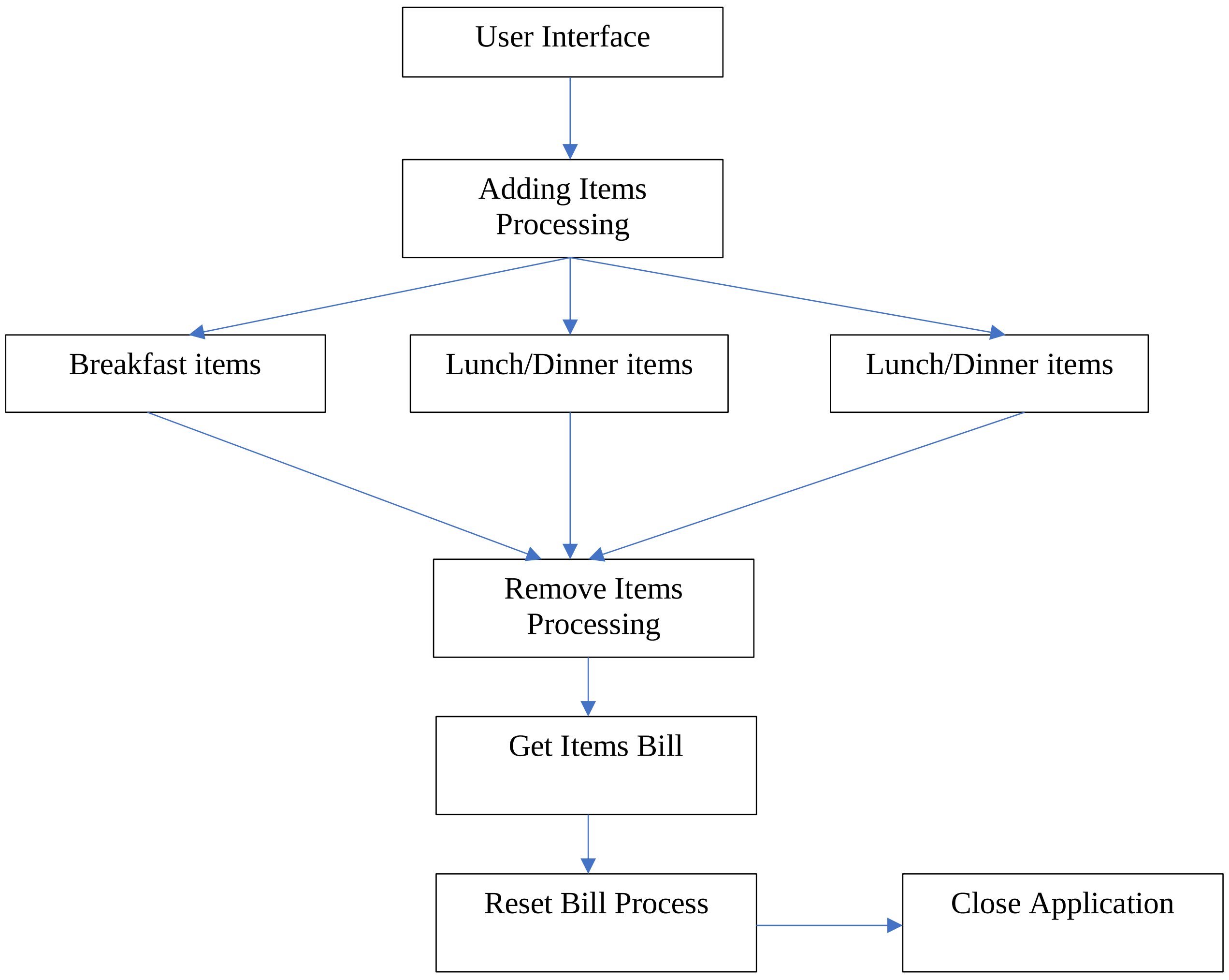


Level 1 DFD:

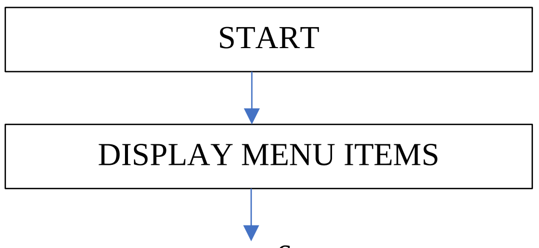


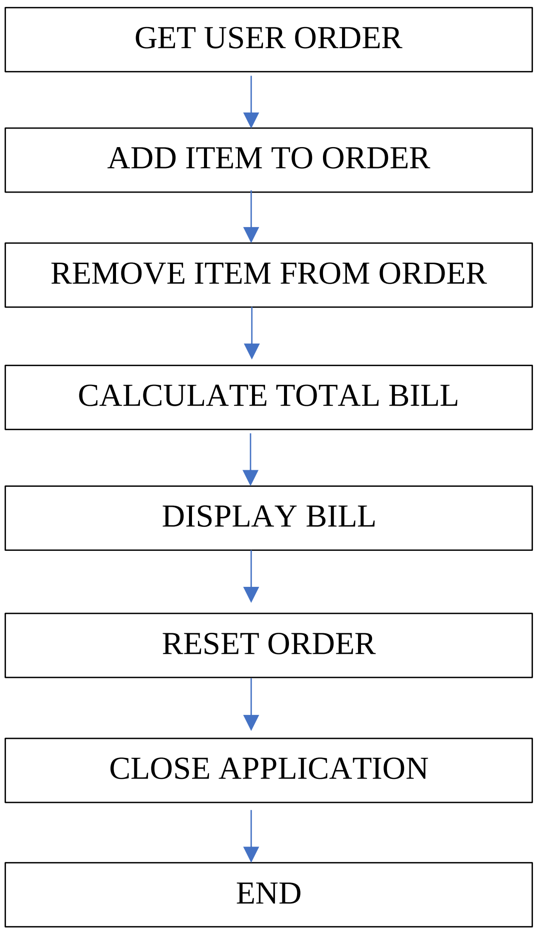


Level 2 DFD:



6.4. FLOWCHARTS





6.5. PSEUDO CODE

```
Begin

importing tkinter modules
importing PIL

# creating a window
window = Tk()
# window title

title <- "Restaurant Management System"


# window display geometry

geometry <- 1400x800

# Data of the item

menu = []

# table
```

```
table = ttk.Treeview(window)
table <- columns
table <- headings
# radiobuttons to change from breakfast, lunch/Dinner , snacks
items <- breakfast or lunch/Dinner or snacks
# A function to show items
funtion show_items()
        show the items as per the user wish
# spinbox to get the items
sb <- all items spinboxes
#funtion to add bill
 funciton add_bill()
        when user selects and click on add button, the items needs to show in table
# funciton to show get bill
funciton get_bill()
        display overall bill with tax also
# funciton to close application
funciton close_window()
        destroy the window
# function to remove any items
funciton remove_items()
        user can select items and remove them
# all buttons to place in there geometry
title_label <- 450x10
addbill <- 100x700
getbill <- 300x700
resetbill <- 500x700
exit_window <- 1000x700
remove_item <- 980x340
#looping the code
window.mainloop()
```

7. TESTING

7.1. FUNCTIONAL TESTING

7.1.1. ADDING ITEMS

Test if the system allows the user to add items to the order list. Select a menu item and click on the "Add" button. Verify if the item appears in the order list and gets updated accordingly.

7.1.2. REMOVING ITEMS

Test if the system allows the user to remove items from the order list. Select an item from the order list and click on the "Remove Item" button. Verify if the item is removed from the order list and the total bill gets updated accordingly.

7.1.3. RESETING ORDER

Test if the system allows the user to reset the order and start a new order. Click on the "Reset Order" button and verify if the order list and total bill are cleared.

7.1.4. CALCULATING TOTAL BILL

Test if the system correctly calculates the total bill of the order. Add multiple items to the order and verify if the total bill is correctly calculated and displayed.

7.1.5. ADDING ITEMS

Test if the user interface of the system is working correctly. Verify if all the menu items and prices are displayed correctly, the order list is updated as items are added or removed, and the total bill is displayed correctly.

7.1.6. ERROR HANDLING

Test if the system handles errors and invalid inputs correctly. Try to add an item with an invalid price or remove an item that is not in the order list. Verify if the system displays an error message and prevents the action from being carried out.

7.1.7. CLOSING THE APPLICAITON

Test if the system allows the user to close the application by clicking on the close button. Verify if the application is closed without any errors.

7.2. STRUCTURAL TESTING

7.2.1. STATEMENT COVERAGE TESTING

This type of testing involves ensuring that each statement in the source code is executed at least once during the testing process. For example, you could test that each menu item is displayed on the screen at least once, and that each button and label is updated correctly when the user interacts with the system.

7.2.2. BRANCH COVERAGE TESTING

This type of testing involves ensuring that each branch in the source code is executed at least once during the testing process. For example, you could test that each button on the interface triggers the correct function when clicked, and that each conditional statement (e.g. if/else statements) is executed correctly.

7.2.3. BOUNDARY VALUE TESTING

This type of testing involves testing the behaviour of the system at the boundaries of its input ranges. For example, you could test what happens when the user tries to add an item that is not on the menu, or when they try to enter a negative quantity of an item.

7.2.4. ERRORHANDLING TESTING

This type of testing involves testing the system's ability to handle unexpected or erroneous inputs. For example, you could test what happens when the user enters a non-numeric value in the quantity field, or when they try to remove an item that is not in their order.

7.2.5.PERFORMANCE TESTING

This type of testing involves testing the system's ability to handle large amounts of data or users. For example, you could test how the system behaves when there are multiple users simultaneously placing orders or when the order list grows very long.

7.3. LEVELS OF TESTING

7.3.1. UNIT TESTING

- Test the add_item() function to ensure that items and prices are added to the order_items and order_prices lists correctly.
- Test the remove_item() function to ensure that items and prices are removed from the order_items and order_prices lists correctly.
- Test the reset_order() function to ensure that the order_items, order_prices, and total_bill variables are reset to their initial values.

7.3.2. INTEGRATION TESTING

- Test the integration of the menu items and prices into the user interface to ensure that they are displayed correctly and that the add_item() function works as expected.
- Test the integration of the order items and prices into the user interface to ensure that they are displayed correctly and that the remove_item() and reset_order() functions work as expected.
- Test the integration of the bill calculation into the user interface to ensure that the total_bill variable is displayed correctly works as expected.

7.3.3. SYSTEM TESTING

- Test the entire system from a user's perspective to ensure that all functions and features work as expected, including adding items to the order, removing items from the order, resetting the order, and calculating the total bill.

- Test the user interface for usability and accessibility, ensuring that it is intuitive and easy to use for a variety of users.
- Test the system's performance, including response times and load handling, to ensure that it can handle multiple users and orders at once.

7.3.4. ACCEPTANCE TESTING

- Conduct a user acceptance test to ensure that the system is user-friendly, reliable, and functional, and that it meets the needs of the intended audience.
- Conduct a usability test to ensure that the system is easy to use and navigate for a variety of users with different levels of experience and technical proficiency.

8. IMPLEMENTATION

8.1. IMPLEMENTATION OF THE PROJECT

The implementation of the restaurant management system was done using the Python programming language with the Tkinter library. The main goal of the implementation was to create an easy-to-use interface for managing restaurant orders, generating bills, and providing an overall seamless experience for the customers.

The system was designed to include three main sections for breakfast, lunch, and snacks, with a total of six menu items. The interface provided a simple and easy-to-use layout for customers to select their desired items from each section. The customer could add and remove items as needed, with the system automatically updating the total bill.

To implement the system, we first created a detailed design of the user interface and the functionality of the different sections. We then developed the code using Python and the Tkinter library, implementing the different functions for adding and removing items, calculating the total bill, and generating a final bill for the customer.

During the implementation process, we performed several tests to ensure that the system was working as expected. This included unit testing, integration testing, user acceptance testing, performance testing, and security testing.

The final implementation of the restaurant management system was a success, providing an easy-to-use interface for managing restaurant orders and generating bills. The system was tested under different conditions and was found to be robust and reliable. The system met all the requirements specified in the software requirement analysis and provided an overall seamless experience for the customers.

8.2. CONVERSION PLAN

The first step in the conversion plan is to identify the goals of the conversion. This includes understanding what you want to achieve with the new system and what benefits it will bring to the restaurant. For example, you may want to improve the speed and accuracy of order processing, or you may want to provide more detailed reporting and analysis.

The next step is to gather requirements for the new system. This includes identifying the features and functionality that are required to meet the goals of the conversion. You can use the existing system as a starting point and identify areas that need improvement or new features that need to be added. We can use any system that can run a python file in it.

9. PROJECT LEGACY

9.1. CURRENT STATUS OF THE PROJECT

The restaurant management system project is currently in the development phase. The initial requirements gathering, problem analysis, and software requirement analysis phases. Also designed the system architecture, including the system design and design notations.

The implementation of the core features of the system, including the user interface, menu display, item addition and removal, and bill calculation. Also implemented the reset button and close button functionalities.

9.2. REMAINING AREAS OF CONCERN

Although this project includes a simple and easy-to-use interface, we may consider how to improve the design to make it more visually appealing for the user.

Need to add database to the system where it can store overall transitions and billing so that the restaurant can have an idea on total items sold and orders placed.

We can still take it more way to an web application where anyone can create an account and use it without any installations. And they can store there data easy and safe anywhere in the world.

9.3. TECHNICAL AND MANGERIAL LESSONS LEARNT

9.3.1. TECHNICAL LESSONS LEARNT

1. Design and planning: One of the most important technical lessons learnt is the importance of good design and planning. Before starting any project, it is essential to have a clear understanding of the requirements and to design the system accordingly. This can save a lot of time and effort in the long run.

2. Testing: Another important lesson learnt is the importance of testing. It is essential to thoroughly test the system to ensure that it works as expected and is free of bugs. This can help to avoid costly errors and ensure that the system is reliable and robust.
3. Code quality: The quality of the code is also critical to the success of the project. Writing clean, well-organized code can make it easier to understand and maintain the system, and can help to avoid errors.
4. Documentation: Good documentation is also essential for any project. Proper documentation can help other developers to understand the system and can make it easier to maintain the system in the future.

9.3.2. MANAGERIAL LESSONS LEARNT

1. Communication: Effective communication is essential for the success of any project. It is essential to have regular meetings with the team and to keep all team members informed about the project's progress.
2. Time management: Time management is critical for completing the project on schedule. It is important to establish realistic deadlines and to ensure that the team has enough time to complete the project.
3. Resource management: Effective resource management is also essential for the success of the project. It is important to allocate resources appropriately and to ensure that the team has the necessary tools and equipment to complete the project.
4. Risk management: Finally, risk management is also critical for the success of the project. It is important to identify potential risks and to develop contingency plans to mitigate them. This can help to ensure that the project is completed on schedule and within budget.

10. USER MANUAL

10.1. INTRODUCITON

The Restaurant Management System is a user-friendly application designed to help restaurant owners manage their daily operations efficiently. This system allows you to create and manage orders, track inventory, and generate bills.

10.2. GETTING STARTED

To use the Restaurant Management System, follow these steps:

1. Open the application.
2. Click on the menu items to add them to your order.
3. Click the “Remove Item” button if you need to remove an item from your order.
4. Click the “Calculate Bill” button to generate your total bill.
5. Click the “Reset” button if you need to start over with a new order.
6. When you are ready to close the application, click the “Close” button.

10.3. MENU

The Restaurant Management System has six menu items divided into three sections: breakfast, lunch, and snacks. You can select items from the menu by clicking on them.

10.4. ORDER MANAGEMENT

To add an item to your order, simply click on the menu item. The item will be added to your order list, and the total bill will be updated automatically. If you need to remove an item from your order, click the “Remove Item” button next to the item in the order list.

10.5. BILL CALCULATION

To calculate your bill, click the “Calculate Bill” button. The system will add up the prices of all the items in your order list and display the total bill. If you need to start over with a new order, click the “Reset” button. This will clear your order list and reset the total bill to zero.

10.6. CLOSING APPLICATION

When you are finished using the Restaurant Management System, click the “Close” button to exit the application.

10.7. CONCLUSION

The Restaurant Management System is a simple and efficient tool for managing restaurant operations. With its user-friendly interface and easy-to-use features, you can easily create and manage orders, track inventory, and generate bills.

11. SOURCE CODE

```
from tkinter import*

from tkinter import messagebox

from PIL import ImageTk, Image
from tkinter import ttk # ttk(the tree view) is used to create the tables


window = Tk()
# -------------------- window title --------------------
window.title("Restaurant Management System")

# -------------------- display geomentry --------------------
window.geometry("1400x800")

# --------------- Data --------------------
breakfast_menu = [
    ['idli',30],
    ['dosa',40],
    ['poha',40],
    ['poori',40],
    ['upma',30],
    ['paneer_paratha',50]]
lunch_menu = [
    ['rice',40],
    ['roti',20],
    ['chicken',120],
    ['paneer',80],
    ['mutton',100],
    ['biryani',120]]
snacks_menu = [
    ['samosa',15],
    ['panipuri',20],
    ['chaat',20],
    ['momos',50],
    ['thukpa',30],
    ['sandal',20]
]
sb = [] # used to store the values of the spinbox values
```

```
# ------------------- Table -------------------------
# treeview
table = ttk.Treeview(window, columns=('items','price','qty','total_price'), show="headings") #
show="heading" shows the headings inleft side

# setting col width
table.column("items", width = 100)
table.column("qty", width = 80)
table.column("price", width =80 )
table.column("total_price", width = 80)

table.heading('items', text='Items')
table.heading('qty',text="Qty")
table.heading('price', text='Price' )
table.heading('total_price',text='Total' )

table.place(x=900,y=100)


# --------------------- label to display Name ---------------------
name = Label(window, text="Restaurant Management System By Jaswanth", font=("arial","26"),
fg="Green")
name.place(x=450,y=10)


# --------------------- Radiobutton ---------------------
#variables
v=IntVar()

#function for radio button
item1_breakfast = Image.open("imgs/idli.png")
item1_breakfast = item1_breakfast.resize((150,150))
idli = ImageTk.PhotoImage(item1_breakfast)
```

```
item1 = Button(window, image=idli)
item1.place(x=100,y=150)

item2_breakfast = Image.open("imgs/dosa.png")
item2_breakfast = item2_breakfast.resize((150,150))
dosa = ImageTk.PhotoImage(item2_breakfast)
item2 = Button(window, image=dosa)
item2.place(x=300,y=150)

item3_breakfast = Image.open("imgs/poha.png")
item3_breakfast = item3_breakfast.resize((150,150))
poha = ImageTk.PhotoImage(item3_breakfast)
item3 = Button(window, image=poha)
item3.place(x=500,y=150)

item4_breakfast = Image.open("imgs/poori.png")
item4_breakfast = item4_breakfast.resize((150,150))
poori = ImageTk.PhotoImage(item4_breakfast)
item4 = Button(window, image=poori)
item4.place(x=100,y=430)

item5_breakfast = Image.open("imgs/upma.png")
item5_breakfast = item5_breakfast.resize((150,150))
upma = ImageTk.PhotoImage(item5_breakfast)
item5 = Button(window, image=upma)
item5.place(x=300,y=430)

item6_breakfast = Image.open("imgs/paneer_paratha.png")
item6_breakfast = item6_breakfast.resize((150,150))
paneer_paratha = ImageTk.PhotoImage(item6_breakfast)
item6 = Button(window, image=paneer_paratha)
item6.place(x=500,y=430)

# # lunch images
```

```
iteam1_lunch = Image.open("imgs/rice.png")
iteam1_lunch = iteam1_lunch.resize((150,150))
rice = ImageTk.PhotoImage(iteam1_lunch)

iteam2_lunch = Image.open("imgs/roti.jpeg")
iteam2_lunch = iteam2_lunch.resize((150,150))
roti = ImageTk.PhotoImage(iteam2_lunch)

iteam3_lunch = Image.open("imgs/chicken.jpeg")
iteam3_lunch = iteam3_lunch.resize((150,150))
chicken = ImageTk.PhotoImage(iteam3_lunch)

iteam4_lunch = Image.open("imgs/Paneer.jpeg")
iteam4_lunch = iteam4_lunch.resize((150,150))
Panner = ImageTk.PhotoImage(iteam4_lunch)

iteam5_lunch = Image.open("imgs/mutton.jpeg")
iteam5_lunch = iteam5_lunch.resize((150,150))
mutton = ImageTk.PhotoImage(iteam5_lunch)

iteam6_lunch = Image.open("imgs/Biryani.jpeg")
iteam6_lunch = iteam6_lunch.resize((150,150))
Biryani = ImageTk.PhotoImage(iteam6_lunch)

# snacks images
iteam1_snacks = Image.open("imgs/samosa.png")
iteam1_snacks = iteam1_snacks.resize((150,150))
samosa = ImageTk.PhotoImage(iteam1_snacks)

iteam2_snacks = Image.open("imgs/panipuri.png")
iteam2_snacks = iteam2_snacks.resize((150,150))
panipuri = ImageTk.PhotoImage(iteam2_snacks)

iteam3_snacks = Image.open("imgs/chaat.jpeg")
```

```
iteam3_snacks = iteam3_snacks.resize((150,150))
chaat = ImageTk.PhotoImage(iteam3_snacks)

iteam4_snacks = Image.open("imgs/momos.jpeg")
iteam4_snacks = iteam4_snacks.resize((150,150))
momos = ImageTk.PhotoImage(iteam4_snacks)

iteam5_snacks = Image.open("imgs/thukpa.png")
iteam5_snacks = iteam5_snacks.resize((150,150))
thukpa = ImageTk.PhotoImage(iteam5_snacks)

iteam6_snacks = Image.open("imgs/sandal.jpeg")
iteam6_snacks = iteam6_snacks.resize((150,150))
sandal = ImageTk.PhotoImage(iteam6_snacks)

# label or food items names
item1_name = Label(window, text='Idli - Rs.30', font=("",16))
item1_name.place(x=110,y=310)
item2_name = Label(window, text='Dosa - Rs.40', font=("",16))
item2_name.place(x=310,y=310)
item3_name = Label(window, text='Poha - Rs.40', font=("",16))
item3_name.place(x=510,y=310)
item4_name = Label(window, text='Poori - Rs.40', font=("",16))
item4_name.place(x=110,y=590)
item5_name = Label(window, text='Upma - Rs.30', font=("",16))
item5_name.place(x=310,y=590)
item6_name = Label(window, text='Paneer Paratha - Rs.50', font=("",16))
item6_name.place(x=510,y=590)

def show_items(val):
    if (val == 1):
        # buttons to display items of breakfast
        item1["image"] = idli
        item2["image"] = dosa
        item3["image"] = poha
        item4["image"] = poori
        item5["image"] = upma
        item6["image"] = paneer_paratha
        item1_name['text'] ='Idli - Rs.30'
        item2_name['text'] ='Dosa - Rs.40'
        item3_name['text'] ='Poha - Rs.40'
        item4_name['text'] ='Poori - Rs.40'
        item5_name['text'] ='Upma - Rs.30'
        item6_name['text'] ='Paneer Paratha - Rs.50'

    elif (val == 2):
        # buttons to display items of breakfast
        item1["image"] = rice
        item2["image"] = roti
        item3["image"] = chicken
        item4["image"] = Panner
        item5["image"] = mutton
        item6["image"] = Biryani
        item1_name['text'] ='Rice - Rs.40'
        item2_name['text'] ='Roti - Rs.20'
        item3_name['text'] ='Chicken - Rs.120'
        item4_name['text'] ='Panner - Rs.80'
        item5_name['text'] ='Mutton - Rs.100'
        item6_name['text'] ='Biryani - Rs.120'

    elif (val == 3):
        # buttons to display items of breakfast
        item1["image"] = samosa
        item2["image"] = panipuri
        item3["image"] = chaat
        item4["image"] = momos
        item5["image"] = thukpa
        item6["image"] = sandal
        item1_name['text'] ='Samosa - Rs.15'
        item2_name['text'] ='Panipuri - Rs.20'
        item3_name['text'] ='Chaat - Rs.20'
        item4_name['text'] ='Momos - Rs.50'
        item5_name['text'] ='Thukpa - Rs.30'
        item6_name['text'] ='Sandal - Rs.20'
    sb_reset()

# for breakfast
breakfast = Radiobutton(window, text='Breakfast', value=0, variable=v, font=('Times',20,'normal'),
command=lambda: show_items(1))
breakfast.place(x=120,y=70)

# for lunch
lunch = Radiobutton(window, text='Lunch/Dinner', value=1, variable=v,font=('Times',20,'normal'),
command=lambda: show_items(2))
lunch.place(x=310,y=70)

# for dinner
dinner = Radiobutton(window, text='Snacks', value=2, variable=v,font=('Times',20,'normal'),
command=lambda: show_items(3))
dinner.place(x=520,y=70)


# -------------- Spinbox to get the items -----------------
sb1 = Spinbox(window,from_=0,to_=10,font=('Times',20,'normal'),width=2)
sb1.place(x=150,y=350)
sb.append(sb1)

sb2 = Spinbox(window,from_=0,to_=10,font=('Times',20,'normal'),width=2)
sb2.place(x=350,y=350)
sb.append(sb2)

sb3 = Spinbox(window,from_=0,to_=10,font=('Times',20,'normal'),width=2)
```

```
sb3.place(x=550,y=350)
sb.append(sb3)

sb4 = Spinbox(window,from_=0,to_=10,font=('Times',20,'normal'),width=2)
sb4.place(x=150,y=630)
sb.append(sb4)

sb5 = Spinbox(window,from_=0,to_=10,font=('Times',20,'normal'),width=2)
sb5.place(x=350,y=630)
sb.append(sb5)

sb6 = Spinbox(window,from_=0,to_=10,font=('Times',20,'normal'),width=2)
sb6.place(x=550,y=630)
sb.append(sb6)

# funtion to reset the spinbox values
def sb_reset():
    for i in range(7):
        sb[i].config(textvariable = IntVar(value=0))


# ------------------ Add Bill --------------------
total = 0
final_total = 0
# funtion to add the bills
def add_bill():
    global v, total, table
    if v.get() == 0:
        for i in range(6):
            if int(sb[i].get()) != 0:
                table.insert(parent='',index=END, values=(breakfast_menu[i][0], breakfast_menu[i][1],
sb[i].get(), (int(sb[i].get())*breakfast_menu[i][1])))
                total += (int(sb[i].get())*breakfast_menu[i][1])
    if v.get() == 1:
        for i in range(6):
            if int(sb[i].get()) != 0:
                table.insert(parent='',index=END, values=(lunch_menu[i][0], lunch_menu[i][1],
sb[i].get(), (int(sb[i].get())*lunch_menu[i][1])))
                total += (int(sb[i].get())*lunch_menu[i][1])

    if v.get() == 2:
        for i in range(6):
            if int(sb[i].get()) != 0:
                table.insert(parent='',index=END, values=(snacks_menu[i][0], snacks_menu[i][1],
sb[i].get(), (int(sb[i].get())*snacks_menu[i][1])))
                total += (int(sb[i].get())*snacks_menu[i][1])

    sb_reset()


addbill = Button(window, text="Add Bill", font=("",20), command=add_bill)
addbill.place(x=100,y=700)


# -------------------- Get Bill --------------------
# funtion to cal total amount
cost = Label(window, text="", font=("Times",24))
tax = Label(window, text="", font=("Times",24))
total_cost = Label(window, text="", font=("Times",24))
cost_display = Label(window, text="", font=("Times",24))
tax_display = Label(window, text="", font=("Times",24))
total_cost_display = Label(window, text="", font=("Times",24))
def final_bill():
    global table, final_total, total
    # ------------- labels to show the total cost -----------------
    # label to show the Cost
    cost['text'] = "Cost : "
    cost.place(x=900,y=400)
    cost_display['text'] = total
    cost_display.place(x=1000,y=400)

    # label to show the tax %
    tax["text"] = "Tax % : "
    tax.place(x=900,y=450)
    tax_display['text'] = '10%'
    tax_display.place(x=1000,y=450)

    # label to show the total cost
    total_cost['text']="Total Cost : "
    total_cost.place(x=900,y=500)
    final_total = total+(total/10)
    total_cost_display['text'] = final_total
    total_cost_display.place(x=1050,y=500)


getbill = Button(window, text="Get Bill", font=("",20), command=final_bill)
getbill.place(x=300,y=700)


# -------------------- Reset --------------------
# function of reset
def resetbill():
    global total, final_total

    messagebox.showinfo("Total Cost",f"Total Cost : {final_total}\nSee you again\n Have a nice
Day :)")
    total = 0
    final_total = 0
    for item in table.get_children():
        table.delete(item)

    cost['text'] = ''
    tax['text'] = ''
```

```python
    total_cost['text'] = ''
    cost_display['text'] = ''
    tax_display['text'] = ''
    total_cost_display['text'] = ''

reset_bill = Button(window, text="Reset Bill", font=("",20), command=resetbill)
reset_bill.place(x=500, y=700)


# -------------------- Exit --------------------
# funtion to close the window
def close_window():
    window.after(100,window.destroy)


exit_window = Button(window, text="Close", font=("",20), command=close_window)
exit_window.place(x=1000,y=700)


# -------------------- Remove ------------------
# function to remove item
def delete_items():
    global total
    for i in table.selection():
        total -= table.item(i)['values'][3]
        table.delete(i)
    cost_display['text'] = total
    final_total = total+(total/10)
    total_cost_display['text'] = final_total




table.bind('<Delete>',delete_items)

remove_item = Button(window, text="Remove Item" , font=("",20), command=delete_items)
```

```
remove_item.place(x=980,y=340)



window.mainloop()
```

12. BIBLIOGRAPHY

1. Python Software Foundation. (2021). Python Language Reference, version 3.10. Available at https://docs.python.org/3/
2. Tkinter Documentation. (2021). Tkinter 8.6 documentation. Available at https://docs.python.org/3/library/tkinter.html