COVER PAGE HAS EBEN DELETED

Table of Contents

[Introduction And Assumptions 3](#_Toc161610265)

[Introduction: 3](#_Toc161610266)

[Assumptions: 3](#_Toc161610267)

[Program design, pseudocode, and explanation 4](#_Toc161610268)

[The first file “user management” 4](#_Toc161610269)

[user management pseudo code 4](#_Toc161610270)

[User Management Source Code And Explanation 9](#_Toc161610271)

[The output of this code 15](#_Toc161610272)

[Now the second file “menu\_table\_management” 18](#_Toc161610273)

[Pseudo code for menu\_table\_management 18](#_Toc161610274)

[Menu Table Management Source Code And Explanation 21](#_Toc161610275)

[The output of this code 24](#_Toc161610276)

[Third file “waiter\_services” 27](#_Toc161610277)

[waiter\_services pseudo code 27](#_Toc161610278)

[Waiter services Source Code And Explanation 38](#_Toc161610279)

[The output of this code 45](#_Toc161610280)

[The fourth file “chef services” 49](#_Toc161610281)

[Chef services pseudo code 49](#_Toc161610282)

[Chef services Source Code And Explanation 53](#_Toc161610283)

[The Output of the code 55](#_Toc161610284)

[The fifth file “ casher” 57](#_Toc161610285)

[Pseudo-code for cashier file 57](#_Toc161610286)

[Cashier Service Source Code And Explanation 64](#_Toc161610287)

[The Output of the code 67](#_Toc161610288)

[Conclusion: 69](#_Toc161610289)

[Referencing 70](#_Toc161610290)

# **Introduction And Assumptions**

## Introduction:

RPOS (Restaurant Point of Sales System) has been uniquely designed to simplify restaurant operations with a complete system able to function as a central mechanism used to maintain transactional efficiency. The aim of preparing this documentation is to describe both the design procedure, implementation, and the different features of the RPOSS system that cater to the unique requirements of the restaurant managers, waiting staff, chefs, and cashiers.

## Assumptions:

User Roles and Responsibilities: The system acknowledges the users will each be assigned specific roles whose responsibilities are defined according to their role—managers, waiting staff, chefs for instance, and cashiers.

Modular Design: Being a modular programming system-based system, it is scalable and maintainable. For every module, the component plays a specific role.

Data Storage: All non-raw data relevant to RPOSS will be saved in text files. The system will utilize the keyword data structure (dictionary, list & CSV) for better data management.   
Menu and Table Management: The system assumes that managers will have to be responsible for updating changes on the food menu items, prices, and tables. The server will be able to dole out tables, record orders and update the waiting list after contacting the manager who provides real-time updates on available tables and other things.

Food Preparation: Chefs can manage their assigned cooking tasks through a dedicated interface, update the status of orders, and check inventory in real time. Billing and Payment: Cashiers will calculate bills for customers, based on the bills, including service charges and taxes.

Validation and Error Handling: The system will connected with verification blocks to maintain correct user inputs and logical errors, improving the program system reliability.

Additional Features: Though the interior detailedness is defined in the system specification several major options to the set can be added in order to improve the consumer experience and optimize efficiency.

User Interface: The system will respond as directed, without any special attention to the graphical user interface, but only exceptionally.

# **Program design, pseudocode, and explanation**

## The first file “user management”

### user management pseudo code

1. Import menu\_table\_management as mtm

2. Import waiter\_services

3. Import chef\_services

4. Import cashier\_services

5. Function authenticate\_user()

6. Prompt user to input username and password

7. Try

8. Open 'data/users.txt' file for reading

9. Read each line in the file

10. Split the line into user\_id, username, role, and password

11. If username and password match any user in the file

12. Return the role of the authenticated user

13. Handle FileNotFoundError

14. Print "User data file not found."

15. Return None if authentication fails

16. End Function

17. Function show\_all\_users()

18. Try

19. Open 'data/users.txt' file for reading

20. For each line in the file

21. Split the line into user\_id, username, role, and password

22. Print user\_id, username, and role

23. Handle FileNotFoundError

24. Print "Users file not found."

25. End Function

26. Function read\_users()

27. Initialize an empty list called users

28. Try

29. Open 'data/users.txt' file for reading

30. For each line in the file

31. Split the line into user\_id, username, role, and password

32. Append a dictionary containing user details to the users list

33. Handle FileNotFoundError

34. Print "User file not found."

35. Return the list of users

36. End Function

37. Function write\_users(users)

38. Open 'data/users.txt' file for writing

39. For each user in the users list

40. Write user details to the file in the

41. format "user\_id, username, role, password"

42. End Function

43. Function create\_user()

44. Prompt user to input user ID, username, role, and password

45. Read existing users from file using read\_users()

46. Append new user details to the list of users

47. Write updated users list to file using write\_users()

48. Print "User created successfully."

49. End Function

50. Function assign\_user\_role()

51. Prompt user to input user ID and new role

52. Read existing users from file using read\_users()

53. Find the user with the specified ID

54. If user is found

55. Update user's role with the new role

56. Write updated users list to file using write\_users()

57. Print "User role updated successfully."

58. Otherwise, print "User not found."

59. End Function

60. Function update\_user\_details()

61. Prompt user to input user ID and new username

62. Read existing users from file using read\_users()

63. Find the user with the specified ID

64. If user is found

65. Update user's username with the new username

66. Write updated users list to file using write\_users()

67. Print "User details updated successfully."

68. Otherwise, print "User not found."

69. End Function

70. Function change\_password()

71. Prompt user to input user ID and new password

72. Read existing users from file using read\_users()

73. Find the user with the specified ID

74. If user is found

75. Update user's password with the new password

76. Write updated users list to file using write\_users()

77. Print "Password changed successfully."

78. Otherwise, print "User not found."

79. End Function

80. Function user\_management\_menu()

81. Repeat until user chooses to return to main menu

82. Print menu options for user management

83. Prompt user to input choice

84. Based on user's choice, execute corresponding function

85. End Function

86. Function manager\_interface()

87. Repeat until user chooses to return to main menu

88. Print menu options for menu and table management

89. Prompt user to input choice

90. Based on user's choice, execute corresponding function

91. End Function

92. Function main\_menu()

93. Repeat until user chooses to exit

94. Print main menu options

95. Prompt user to input choice

96. Based on user's choice, execute corresponding function

97. End Function

98. Function main()

99. Authenticate user

100. If authentication is successful

101. If user role is manager

102. Display main menu

103. Else if user role is chef

104. Display chef interface

105. Else if user role is cashier

106. Display cashier interface

107. Else if user role is waiter

108. Display waiter interface

109. Else

110. Print "Unknown role. Exiting."

111. End Function

112. Main program execution

113. Call main() function

### User Management Source Code And Explanation

A screen shot of a computer program

Description automatically generated

**authenticate\_user():**

This is the point where the user must provide the username and password to proceed. After that, the code tries to launch the 'data/users.txt' file for reading. Every line of the file is split by the user\_id, title, role and password. If the checker with the given username and password any user in the file and it returns the identity of the authorized user. If the file is not found it will say "User data file not available." If authentication doesn't succeed, it yields None.

**show\_all\_users():**

The method is trying to open a file named 'data/users.txt', so it can be read. It parses each line in the file and splits the line to user\_id, username, role and password. It further outputs user\_id, username and role for each user. If file is not available it will print "No users file found"

A computer screen with many colorful text

Description automatically generated

**read\_users():**

It will therefore assign an empty list to an object of the type User List, which is users here. It attempts to open the users' information file named 'data/users.txt' for reading. Subsequently, it parses every line; and as a result, it splits lines into user id, username, roles, and passwords, and the dictionary of user information is appended to the users list. If the file is not found, it provides "User file not found." output message. The command ends by calling a function that displays the list of users as well.

**write\_users():**

This working of the function calls for the writing of data in the 'data/users.txt' file. For every user in users list (as an x) it puts user details into the file with the format "user\_id, username, role, password" (as a y).

**create\_user():**

By prompting the user to enter their user ID, username, role, and password, this function allows the user to validate their authorization status. It utilizes read\_users() function to populate the users it already has from the file. It uses the append() function in the array class to insert the data of users and then calls the write\_users() function to write the newly updated list of users to the file. In the end, its code reads, "User created successfully."

**assign\_user\_role():**

Then user is prompted to enter the user ID and new role in the role adding function. It reads users, which are in a file, using the read\_users() function. It returns the data on the user of these matching ID. If the user is found, it will make the user's history be shown for the new role, write the updated users list into the file by using the write\_users() function and the user roles are updated successfully message is printed. The printer eventually says "User not found." if a person does not exist, instead.

A screenshot of a computer screen

Description automatically generated

**update\_user\_details():**

This function lets the user input user id and a new username through it. It receive user list which already exists from the file using the read\_users() method. It gets the information specified for the user with a matching ID. When there is a user entry it first updates the username to the new one, and finally writes the updated users list to the file with the help of the write\_users() function and prints “User details updated successfully.” If not, it will request User not found.

**change\_password():**

This implementation causes a program to ask the user to kindly provide his or her user ID and his or her new password. It reads bunch of existing users from the file using the function called read\_users(). Here a unique number of the user it's looking for gets found. In case of a successful search, it rewrites the user's password to the new password, it invokes the write\_users() function to write the updated list of users to the file containing the 'users.txt' file name, and it prints "Password updated successfully". Otherways, it outputs: "User not found."

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**user\_management\_menu():**

It is responsible for issuing a custom menu for user management functions. It repeatedly tells the user to choose options before he will be allowed to go back to the main menu. According to the usage option, it acts out that particular task the user selects.

**main\_menu():**

That functionality is this menu's principal role. It continues with this pattern until it is ended. It asks the user again and again until they select an option to move on their own. The execution of the chosen function is why, according to the user’s decision, it does it.

A screenshot of a computer

Description automatically generated

A black rectangle with white text

Description automatically generated

**manager\_interface():**

This function is a menu for table/ menus management operation in the database. A repetitive request to put in the option will be as such followed until user inputs ‘return’ into the main menu. Implementing the chosen command, specifically the corresponding function.

**main():**

This specifically is called the entrance of the program. The authentication system verifies user identity where applicable and leads the user to the respective interface depending on their job title. This could be the main menu, the chef’s menu, the cashier’s menu, or the waiter’s menu. After getting the role if it is not known to the user, it prints "Unknown role. Exiting."

**Main program execution:**

The assembly directive links the program to the start with the main() function.

### The output of this code

**authenticate\_user choice**

Now I entered the correct manager username and password and this is the output.

A black background with white dots

Description automatically generated

Now I will try logging in with the wrong username and password.



**Now in user management show\_all\_users choice**

It will print all the users in the file as shown in the picture below.

A screenshot of a computer

Description automatically generated

**Now let’s try the Create user choice**

as you can see in the picture below I created a new user called “test” and is saved successfully in the file users.txt

**A screenshot of a computer

Description automatically generated**

**Update user role choice**

So here as you can see I managed to change the user role from ”manager” to “ waiter” and it also saved successfully in the users.txt file

A screenshot of a computer

Description automatically generated

**Update user details choice**

Using this function I can change any username from the users.txt file by the user id here I changed the username from “test” to “new\_test”

A screenshot of a computer

Description automatically generated

**Change password choice**

with this function, I can change any user password using the id of the user like now I changed the password from “ test” to “password”

A screenshot of a computer

Description automatically generated

**The return to main menu choice** just exits this user management interface it it goes back to the manager interface

## Now the second file “menu\_table\_management”

### Pseudo code for menu\_table\_management

1. Function add\_menu\_item():

2. Prompt user to input menu item ID, name, and price

3. Open 'data/menu.txt' file for appending

4. Write the menu item details to the file in the format "item\_id,name,price"

5. Print "Menu item added successfully."

6. End Function

7. Function update\_menu\_item():

8. Prompt user to input menu item ID to update, new name, and new price

9. Read existing menu items from the file

10. Initialize a variable 'updated' to False

11. Loop through each item in the menu items list

12. If the item ID matches the specified ID

13. Update the item's name and price with the new values

14. Set 'updated' to True

15. If 'updated' is True

16. Open 'data/menu.txt' file for writing

17. Write the updated menu items back to the file

18. Print "Menu item updated successfully."

19. Else

20. Print "Menu item not found."

21. End Function

22. Function add\_table():

23. Prompt user to input a unique table ID and number of pax

24. Set the status to "free" (default status for a new table)

25. Open 'data/tables.txt' file for appending

26. Write the table details to the file in the format "table\_id,pax,status"

27. Print "Table added successfully."

28. End Function

29. Function update\_table():

30. Prompt user to input table ID to update and new number of pax

31. Initialize a variable 'updated' to False

32. Initialize an empty list 'tables'

33. Try

34. Open 'data/tables.txt' file for reading

35. Loop through each line in the file

36. Split the line into parts: table\_id, pax, status

37. If the table ID matches the specified ID

38. Update the number of pax with the new value

39. Set 'updated' to True

40. Append the line (either updated or not) to the 'tables' list

41. If 'updated' is True

42. Open 'data/tables.txt' file for writing

43. Write the updated table details back to the file

44. Print "Table updated successfully."

45. Else

46. Print "Table not found."

47. Except FileNotFoundError

48. Print "Tables file not found."

49. End Function

50. Function show\_available\_tables():

51. Print "Available Tables:"

52. Try

53. Open 'data/tables.txt' file for reading

54. Loop through each line in the file

55. Split the line into parts: table\_id, pax, status

56. If the status is "free"

57. Print the table ID, number of pax, and status

58. Except FileNotFoundError

59. Print "Tables file not found."

60. End Function

61. Function cancel\_block\_order():

62. Prompt user to input menu item ID to cancel/block

63. Read existing menu items from the file

64. Initialize a variable 'blocked' to False

65. Loop through each item in the menu items list

66. If the item ID matches the specified ID

67. Append "blocked" to the item to indicate it's blocked

68. Set 'blocked' to True

69. If 'blocked' is True

70. Open 'data/menu.txt' file for writing

71. Write the updated menu items back to the file

72. Print "Menu item blocked successfully."

73. Else

74. Print "Menu item not found."

75. End Function

### Menu Table Management Source Code And Explanation

A screen shot of a computer program

Description automatically generated

**add\_menu\_item():**

Prompts the user to give item ID, its name, and price. Writes to the 'data/menu.txt' file in append mode. Provide the dish details into the file. Added a menu item: "Menu item added successfully."

**update\_menu\_item():**

Users are required to enter the menu item ID they would be updating, the new name, and the new price. Goes through the list of existing dishes in the current menu. Changes the menu item as specified with the new name and price. The updated menu items should be written back to the file. Prints "Successfully updated the menu item"

**add\_table():**

Starts by getting the user to provide a personalized table ID and number of pax. It sets the flag to "free" (the initial status for the new table). Add new records to the data table in "data/tables.txt". Writes the details of the table to the file. Prints "Table added successfully."

A screen shot of a computer program

Description automatically generated

A screen shot of a computer code

Description automatically generated

**add\_table():**

Directs the user to enter specific table identification and number of pax. Sets the status value to "free" (for "free" - the default table status). Prepends into the file named 'data/tables.txt'. Save the details in the file. Prints "Table added successfully."

**update\_table():**

Query the user for the table ID to update, input the new number of pax. Takes a look at the current table details from the specified file. Updates the given table with the new quantity irrespective of pax. Writes the up-to-date details concerning the file back into the file. Prints "Table updated successfully."

**show\_available\_tables():**

Prints the available tables by reading the 'data/tables.txt' file. Filters and displays only the tables with the status "free."

A screen shot of a computer program

Description automatically generated

**cancel\_block\_order():**

There will be a prompt on the screen requesting the user to key in the menu item ID to be used for cancellation/blocking. Get\_menu\_items from the file. Adds the mentioned menu item to "Block list." Writes the updated menu item listings back to the file. "Menu item blocked successfully - printing."

### The output of this code

**Add menu item choice**

As you can see I managed to add a new item to the menu successfully

A screenshot of a computer

Description automatically generated

**Update menu item choice**

So in this function, by the item ID, I can change its name and price as you can see in the picture below I changed the item “orange” to “fresh orange juice” and the price from “3” to “5” And its updated successfully

A screenshot of a computer

Description automatically generated

**Add table choice**

As you can see I added a new table to the system and its called “B2” and I specified the number of pax to be “4” and this will be saved in the tables.txt file

A screenshot of a computer

Description automatically generated

**Update table choice**

In this function, I will use the table ID to edit it like adding more chars so it can have more pax, and in the picture below I changed the “B2” table to have “6” pax

A screenshot of a computer

Description automatically generated

**Show available tables choice**

This will print all the free tables in the tables.txt any occupied tables won’t be printed



**Cancel or block order choice**

in this function, I can block menu items also by their id I choose to block pizza now pizza is no longer available in the customer menu

A screenshot of a computer

Description automatically generated

**The return to main menu choice** just exits this user management interface it goes back to the manager interface

## Third file “waiter\_services”

### waiter\_services pseudo code

1. Function display\_orders():

2. Print "ready-made orders:"

3. Try

4. Open 'data/orders.txt' file for reading

5. Loop through each line in the file

6. If the line is empty or doesn't contain enough comma-separated values

7. Continue to the next iteration

8. Split the line into parts: order\_id, table\_id, waiter\_id, status, details

9. If status equals "ready\_to\_serve"

10. Print order details

11. Except FileNotFoundError

12. Print "Orders file not found. Please ensure the 'data/orders.txt' file exists."

13. End Function

14. Function generate\_next\_order\_id():

15. Try

16. Open 'data/orders.txt' file for reading

17. Extract order IDs from each line in the file

18. Calculate the next order ID by incrementing the maximum order ID by 1

19. Except FileNotFoundError

20. Set next\_order\_id to 1

21. Return next\_order\_id

22. End Function

23. Function show\_available\_tables():

24. Print "Available Tables:"

25. Try

26. Open 'data/tables.txt' file for reading

27. Loop through each line in the file

28. Split the line into parts: table\_id, pax, status

29. If status equals "free"

30. Print table details

31. Except FileNotFoundError

32. Print "Tables file not found. Please ensure the 'data/tables.txt' file exists."

33. End Function

34. Function is\_valid\_table(table\_id):

35. Try

36. Open 'data/tables.txt' file for reading

37. Loop through each line in the file

38. Split the line into parts: id, \_, \_

39. If id equals table\_id

40. Return True

41. Return False

42. Except FileNotFoundError

43. Print "Tables file not found. Please ensure the 'data/tables.txt' file exists."

44. Return False

45. Return False

46. End Function

47. Function check\_in\_customer():

48. show\_available\_tables()

49. table\_id = input("Enter the table ID to check in customer: ")

50. If not is\_valid\_table(table\_id)

51. Print "Invalid table ID."

52. Return

53. While True:

54. pax\_input = input("Enter number of pax: ")

55. If pax\_input.isdigit() and int(pax\_input) > 0

56. customer\_pax = int(pax\_input)

57. Break

58. Else

59. Print "Invalid input. Please enter a valid number for the number of pax."

60. waiter\_id = input("Enter your Waiter/Waitress ID: ")

61. updated = False

62. tables = []

63. Try

64. Open 'data/tables.txt' file for reading

65. Loop through each line in the file

66. Split the line into parts: table\_id\_file, pax, status

67. If table\_id\_file equals table\_id

68. If status equals "free" and int(pax) >= customer\_pax

69. Append f"{table\_id},{pax},occupied,{waiter\_id}\n" to tables list

70. Set updated to True

71. Else

72. Append line to tables list

73. Else

74. Append line to tables list

75. If updated

76. Open 'data/tables.txt' file for writing

77. Write tables list to the file

78. Print f"Customer checked in at Table {table\_id}."

79. Call take\_order(table\_id, customer\_pax, waiter\_id)

80. Else

81. Print "Unable to check in customer. Please ensure the table is available and can accommodate the pax."

82. Except FileNotFoundError

83. Print "Tables file not found. Please ensure the 'data/tables.txt' file exists."

84. End Function

85. Function read\_menu\_items():

86. menu\_items = []

87. Try

88. Open 'data/menu.txt' file for reading

89. Loop through each line in the file

90. Split the line into parts: \_, item\_name, \_

91. Append item\_name to menu\_items list

92. Except FileNotFoundError

93. Print "Menu file not found. Please ensure the 'data/menu.txt' file exists."

94. Return menu\_items

95. End Function

96. Function read\_menu\_items\_with\_prices():

97. menu\_items = {}

98. Try

99. Open 'data/menu.txt' file for reading

100. Loop through each line in the file

101. Split the line into parts: \_, item\_name, item\_price, \_

102. If item\_price is not "blocked"

103. Add item\_name as key and item\_price as value to menu\_items dictionary

104. Except FileNotFoundError

105. Print "Menu file not found. Please ensure the 'data/menu.txt' file exists."

106. Return menu\_items

107. End Function

108. Function take\_order(table\_id, customer\_pax, waiter\_id):

109. Print f"Taking orders for {customer\_pax} customers at Table {table\_id} served by Waiter/Waitress ID {waiter\_id}."

110. menu\_items\_with\_prices = read\_menu\_items\_with\_prices()

111. orders = []

112. For i in range(customer\_pax):

113. Print "Available menu items:"

114. For item, price in menu\_items\_with\_prices.items():

115. Print f"{item}: {price}"

116. While True:

117. item\_name = input(f"Enter order for customer {i+1}: ")

118. If item\_name in menu\_items\_with\_prices

119. quantity = input(f"Enter quantity for {item\_name}: ")

120. Try

121. quantity = int(quantity)

122. item\_price = float(menu\_items\_with\_prices[item\_name]) \* quantity

123. order\_id = generate\_next\_order\_id()

124. order\_line = f"{order\_id},{table\_id},{waiter\_id},received,{item\_name},{quantity},{item\_price:.2f}\n"

125. Append order\_line to orders list

126. Print f"Added {quantity} of {item\_name} to the order at total price {item\_price:.2f}."

127. Break

128. Except ValueError

129. Print "Invalid quantity. Please enter a numeric value."

130. Else

131. Print "This item is not available. Please choose an available item from the menu."

132. Try

133. Open 'data/orders.txt' file for appending

134. Write orders list to the file

135. Print "Orders successfully taken and recorded."

136. Call chef\_services.chef\_interface()

137. Except Exception as e

138. Print f"Error recording orders: {e}"

139. End Function

140. Function update\_order\_status(order\_id, new\_status):

141. updated = False

142. orders = []

143. Try

144. Open 'data/orders.txt' file for reading

145. Loop through each line in the file

146. If the line is empty or doesn't contain enough comma-separated values

147. Continue to the next iteration

148. Split the line into parts: id, \_, \_, \_, \_, \_, \_

149. If id equals order\_id

150. Append f"{id},{table\_id},{waiter\_id},{new\_status},{item\_name},{quantity},{total\_price}\n" to orders list

151. Set updated to True

152. Else

153. Append line to orders list

154. If updated

155. Open 'data/orders.txt' file for writing

156. Write orders list to the file

157. Print f"Order {order\_id} updated to {new\_status}."

158. Else

159. Print "Order ID not found."

160. Except FileNotFoundError

161. Print "Orders file not found. Please ensure the 'data/orders.txt' file exists."

162. End Function

163. Function update\_order\_status\_to\_served():

164. order\_id = input("Enter the order ID to mark as served: ")

165. Call update\_order\_status(order\_id, "served")

166. End Function

167. Function check\_out\_customer():

168. table\_id = input("Enter the table ID to check out: ")

169. Call update\_table\_status(table\_id, "needs\_cleaning")

170. Print f"Table {table\_id} marked for cleaning."

171. End Function

172. Function clean\_table():

173. table\_id = input("Enter the table ID to mark as clean: ")

174. Call update\_table\_status(table\_id, "free")

175. Print f"Table {table\_id} is now free and clean."

176. End Function

177. Function update\_table\_status(table\_id, new\_status):

178. updated = False

179. tables = []

180. Try

181. Open 'data/tables.txt' file for reading

182. Loop through each line in the file

183. Split the line into parts: id, \_, \_

184. If id equals table\_id

185. If new\_status equals "free"

186. Append f"{id},{pax},{new\_status}\n" to tables list

187. Else

188. Append f"{id},{pax},{new\_status}" to tables list

189. Set updated to True

190. Else

191. Append line to tables list

192. If updated

193. Open 'data/tables.txt' file for writing

194. Write tables list to the file

195. Print f"Table {table\_id} status updated to {new\_status}."

196. Else

197. Print "Table ID not found."

198. Except FileNotFoundError

199. Print "Tables file not found. Please ensure the 'data/tables.txt' file exists."

200. End Function

201. Function is\_table\_free(table\_id):

202. Try

203. Open 'data/tables.txt' file for reading

204. Loop through each line in the file

205. Split the line into parts: id, \_, status

206. If id equals table\_id

207. Return status equals "free"

208. Except FileNotFoundError

209. Print "Tables file not found. Please ensure the 'data/tables.txt' file exists."

210. Return False

211. Return False

212. End Function

213. Function accept\_reservation():

214. While True

215. table\_id = input("Enter table ID for the reservation: ")

216. If is\_table\_free(table\_id)

217. Break

218. Else

219. Print "This table is not available. Please choose another table."

220. Continue\_choice = input("Try another table? (yes/no): ").lower()

221. If continue\_choice is not 'yes'

222. Print "Reservation cancelled."

223. Return

224. customer\_name = input("Enter customer name: ")

225. contact\_number = input("Enter contact number: ")

226. number\_of\_pax = input("Enter number of pax: ")

227. reservation\_date\_time = input("Enter reservation date and time (YYYY-MM-DD HH:MM): ")

228. reservation\_code = generate\_reservation\_code(customer\_name, contact\_number, table\_id, reservation\_date\_time)

229. Try

230. Open 'data/reservations.txt' file for appending

231. Write f"{reservation\_code},{customer\_name},{contact\_number},{number\_of\_pax},{table\_id},{reservation\_date\_time}\n" to the file

232. Print f"Reservation made successfully. Reservation Code: {reservation\_code}"

233. Call block\_table\_for\_reservation(table\_id, reservation\_date\_time)

234. Except FileNotFoundError

235. Print "Error: Reservation could not be made."

236. End Function

237. Function generate\_reservation\_code(customer\_name, contact\_number, table\_id, reservation\_date\_time):

238. Base\_str = f"{customer\_name}{contact\_number}{table\_id}{reservation\_date\_time}"

239. Hash\_object = hashlib.md5(Base\_str.encode())

240. Reservation\_code = Hash\_object.hexdigest()[:6]

241. Return Reservation\_code

242. End Function

243. Function retrieve\_reservation\_by\_code():

244. Reservation\_code = input("Enter the reservation code: ")

245. Found = False

246. Try

247. Open 'data/reservations.txt' file for reading

248. Loop through each line in the file

249. Split the line into parts: code, customer\_name, contact\_number, number\_of\_pax, table\_id, reservation\_date\_time

250. If code equals Reservation\_code

251. Print f"Reservation Found:\n- Customer Name: {customer\_name}\n- Contact Number: {contact\_number}\n- Number of Pax: {number\_of\_pax}\n- Table ID: {table\_id}\n- Reservation Date and Time: {reservation\_date\_time}"

252. Set Found to True

253. Break

254. Except FileNotFoundError

255. Print "Reservations file not found. Please ensure the 'data/reservations.txt' file exists."

256. If not Found

257. Print "No reservation found for the provided code."

258. End Function

259. Function block\_table\_for\_reservation(table\_id, reservation\_date\_time):

260. updated = False

261. tables = []

262. Try

263. Open 'data/tables.txt' file for reading

264. Loop through each line in the file

265. Split the line into parts: id, pax, \_

266. If id equals table\_id

267. Append f"{id},{pax},reserved\n" to tables list

268. Set updated to True

269. Else

270. Append line to tables list

271. If updated

272. Open 'data/tables.txt' file for writing

273. Write tables list to the file

274. Else

275. Print "Table ID not found."

276. Except FileNotFoundError

277. Print "Tables file not found. Please ensure the 'data/tables.txt' file exists."

278. End Function

### Waiter services Source Code And Explanation

A computer screen shot of a program code

Description automatically generated

A computer screen with text

Description automatically generated

**display\_orders():**

Lastly, this function will be responsible for displaying all the orders that are previously received and are ready to be served. It opens the file "orders.txt" and reads the instructions from there. Then, it prints out the information about each order having the status "ready\_to\_serve".

**generate\_next\_order\_id():**

This function returns the next ID value, to be used as the order ID, in "orders.txt" file when the current highest order ID reads that file. The cursor will return an ID of 1 if the file doesn't exist or is empty.

**show\_available\_tables():**

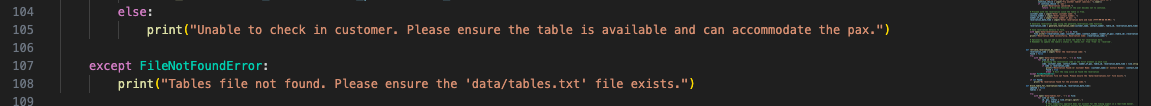
It checks files "tables.txt" for table availability and displays them. It produces the particular tables where its status is "up".

A screenshot of a computer

Description automatically generated

A screen shot of a computer screen

Description automatically generated



**is\_valid\_table(table\_id):**

This function serves the purpose of looking for the table ID in the tables data with the provided one. It performs a simple check from the "tables.txt" table, if the table ID is found then it returns True otherwise it returns False.

**check\_in\_customer():**

This operations include seating a guest to a customer. It prompts the user for the table ID, number of pax, waiter/waitress ID. Then it sets the table status to "occupied" if the table is free and it can contain the number pax's provided.

A screen shot of a computer program

Description automatically generated

A computer screen shot of a program code

Description automatically generated

read\_menu\_items(): This function reads menu items from a file named "menu.txt" and generates a list of item names that returns as a list. It is assumed that in each line in the file, represents the menu item.

read\_menu\_items\_with\_prices(): This function performs the task of reading menu items as well as their prices from the ‘menu.txt’ file and returns a tuple of the two, which is then stored in a dictionary with item names as keys and their respective prices as values.

take\_order(table\_id, customer\_pax, waiter\_id): Guests can place a reservation for a table as another function of this offer. It invites the user to initiate a new order for each customer the quantity of which should also be provided. It reviews the orders first and then records the orders by saving them in the orders.txt file along with details of that order.

A screen shot of a computer program

Description automatically generated

**update\_order\_status(order\_id, new\_status):**

This function determines the current status of the order number being looked for and returns the new status of the order as searched for. The program will be taken from the 'orders.txt' file, and status of the defined order will be updated and the updated orders will be written back to the file.

**update\_order\_status\_to\_served():**

This purpose has an order id as the user is prompted to input. This would invoke the update\_order\_status() function which will update the order status to "served".

**check\_out\_customer():**

This functionality is significant since it indicates a table as being required to be cleaned once the customer is done. It stipulates the user to insert the table ID to be cleared out and change the table status to free.

A screen shot of a computer program

Description automatically generated

A black screen with text

Description automatically generated

**clean\_table():**

This function checks that a table has been properly wiped and considered as being clean. In here, entering the table ID helps the user to mark off the table as cleaned and modifies the status of the table accordingly.

**update\_table\_status(table\_id, new\_status):**

This function implements a status change to the individual table that belongs to a certain ID by replacing the status with the new one. It takes the tables from the "tables.txt" file, updates the specified table status, and writes back replaced tables to the file.

**is\_table\_free(table\_id):**

This function fulfils the purpose of seeing if the specified table, with an id as the key, is busy or not. It reads from the alleys.txt file and returns a True value if the table is free, otherwise returns a False value.

A computer screen shot of a program code

Description automatically generated

A screen shot of a computer program

Description automatically generated

**accept\_reservation():**

This function allows a user to make table reservation. It obliges a user to input details, such as a customer name, contact number, number of the person, and reservation date/time. Lastly, it generates the reservation code, records the reservation detail in the ```reservations.txt``` file and updates the table status to ```reserved```.

**generate\_reservation\_code():**

This function creates a reservation code for the entered customer details and time/date of booking. It generates an MD5 digest of the concatenated strings of these details where first 6 characters of the hexadecimal digest is the returned.

**retrieve\_reservation\_by\_code():**

This function is an accessory that gets reservation information by its reference code. It reads from "reservations.txt", locates the reservation with the code specified, and prints out the reservation details if available.

**block\_table\_for\_reservation():**

This means when a table is requested it updates to "booked" status. Its working is directed to the "tables.txt" file, it updates the status of the given table and writes back the updated tables to the "tables.txt" file.

### The output of this code

**Show available tables choice**

This will print all the free tables in the tables.txt any occupied tables won’t be printed

A screenshot of a computer

Description automatically generated

**Show ready-made orders choice**

This will print all the orders that are ready to be served this allows the waiter to see what food is finished and ready to be served to the customer you can see that is the picture below

A screenshot of a computer

Description automatically generated

**Check-in customers and receive orders choice**

As you can see in the picture below when I wanted to check in customer I only got the available tables I chose one of the available tables now I choose “B5” This table has a capacity of 6 pax but I seated 2 in it then I served them took every one’s order then all this will be saved in a file called order.txt

A screenshot of a computer

Description automatically generated

**Serve order to customer choice**

now after the chef finishes from preparing the food and it is ready to be served, I serve the order to the customer by the order id like shown in the picture below, of course the order status in the orders file will be updated

A screenshot of a computer

Description automatically generated

**Check out customer choice**

I will check out the customer from the table by the table id like shown in the picture below and the table status will change the table needs cleaning

A screenshot of a computer

Description automatically generated

**Clean table choice**

This will also use the table id to clean the table and update it to free

A screenshot of a computer

Description automatically generated

**Accept reservation choice**

this will ask the customer to put his name, phone number, table he wants, date and time like shown in the picture below and he will get a code so when he comes to the restaurant he will give us the code and we will see if he is registered or not



**Show reservation choice**

as I said with the reservation code I can see all the details from here like shown in the picture

A black rectangle with white text

Description automatically generated

Exit will just quit the system

## The fourth file “chef services”

### Chef services pseudo code

1. function display\_orders()

2. print "Received Orders"

3. try

4. open 'data/orders.txt' as file

5. for each line in file

6. parts = split line by comma

7. if length of parts equals 7 and parts[0] is numeric and parts[3] is not 'served'

8. order\_id, table\_id, waiter\_id, status, item\_name, quantity, price = parts

9. print "Order ID", order\_id, "Table ID", table\_id, "Waiter ID", waiter\_id, "Status", status, "Item", item\_name, "Quantity", quantity, "Price", price

10. except FileNotFoundError

11. print "Orders file not found. Please ensure the 'data/orders.txt' file exists."

12. end function

13. function update\_order\_status()

14. order\_id\_to\_update = input("Enter the order ID to update: ")

15. print "Preparing order. This will take 10 seconds..."

16. for remaining in range(10, 0, -1)

17. print remaining, "seconds remaining.", end='\r'

18. sleep for 1 second

19. print "Order is now ready to serve! "

20.

21. new\_status = 'ready\_to\_serve' # Automatically set the new status after delay

22. updated = False

23. updated\_orders = []

24.

25. try

26. open 'data/orders.txt' as file

27. for each line in file

28. parts = split line by comma

29. if parts[0] equals order\_id\_to\_update

30. parts[3] = new\_status # Update the status

31. updated\_line = join parts with comma and append '\n'

32. add updated\_line to updated\_orders

33. updated = True

34. else

35. add line to updated\_orders

36.

37. if updated

38. open 'data/orders.txt' for writing as file

39. write each line in updated\_orders to file

40. print "Order", order\_id\_to\_update, "updated to", new\_status

41. else

42. print "Order ID", order\_id\_to\_update, "not found."

43. except FileNotFoundError

44. print "Orders file not found. Please ensure the 'data/orders.txt' file exists."

45. end function

46. function cancel\_order()

47. order\_id = input("Enter the order ID to cancel/block: ")

48. reason = input("Enter reason for cancellation/blocking: ")

49. updated = False

50. orders = []

51.

52. try

53. open 'data/orders.txt' as file

54. for each line in file

55. parts = split line by comma

56. if length of parts is less than 6

57. print "Skipping line due to unexpected format:", line

58. continue # Skip lines that do not conform to the expected format

59. id, table\_id, waiter\_id, status, item, price = parts

60. if id equals order\_id

61. add "{id},{table\_id},{waiter\_id},cancelled\_due\_to\_{reason},{item},{price}\n" to orders

62. updated = True

63. else

64. add line to orders

65.

66. if updated

67. open 'data/orders.txt' for writing as file

68. write each line in orders to file

69. print "Order", order\_id, "cancelled due to", reason

70. else

71. print "Order ID not found."

72. except FileNotFoundError

73. print "Orders file not found. Please ensure the 'data/orders.txt' file exists."

74. end function

75. function chef\_interface()

76. while True

77. print "\n╔════════════════════════════════════╗"

78. print "║ Chef Interface ║"

79. print "╠════════════════════════════════════╣"

80. print "║ 1. Display Orders ║"

81. print "║────────────────────────────────────║"

82. print "║ 2. Update Order Status ║"

83. print "║────────────────────────────────────║"

84. print "║ 3. Cancel Order ║"

85. print "║────────────────────────────────────║"

86. print "║ 4. Exit ║"

87. print "╚════════════════════════════════════╝"

88.

89. choice = input("\nEnter your choice (1-4): ")

90.

91. if choice equals '1'

92. display\_orders()

93. elif choice equals '2'

94. update\_order\_status()

95. elif choice equals '3'

96. cancel\_order()

97. elif choice equals '4'

98. print "Exiting..."

99. break

100. else

101. print "Invalid choice, please try again."

102. end function

103. if \_\_name\_\_ equals "\_\_main\_\_"

104. chef\_interface()

105. end if

### Chef services Source Code And Explanation

A computer screen shot of colorful text

Description automatically generated

A screen shot of a computer program

Description automatically generated

**display\_orders():**

This function fetches the data from the file named 'data/orders.txt', prints out the details of the order along with the ones that haven't been served yet. In order to ensure that the file has the correct information (order ID, table ID, waiter ID, status, item name, quantity and price), and the status of the order is not 'served', it runs through. If the file hasn't been located, it gives a message containing an error.

**update\_order\_status():**

This method serves to record an order status which is stored in the orders file. It further asks the user to enter the order ID and press 'Update' to place a new order. Finally, it goes into the phase to make an order, which takes a while, to get it ready. Then it automatically pulls the order states and sets it as 'ready\_to\_serve'. If the order ID was not found it will send the user a message of error. In case of the unavailability of the file, an error message is printed.

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

**cancel\_order():** Therefore, this function permits to process cancellation or rejection of an order. It makes the user enter both order ID and the purpose of cancellation/blocking. It amends the status from ‘At risk of being cancelled’ to ‘Cancelled due to reason’ and validates the new order proceeding. If the order ID is not found, an error message is printed by its code. If file does not exist, an error is sent for the message.

**chef\_interface():**a component is developed to link the system with the chef at the kitchen. It shows a list that contains actions such as Orders, Cancel orders, show status of order and exit from the interface. Following the specific input from the user, the function being called is decided. The process will keep on going till the exit option of the user is chosen.

### The Output of the code

The chef’s menu is different because when the waiter takes the order it will directly go to the chef inter face like shown in the picture below so ones the order is received it will open the chef menu

A screenshot of a computer

Description automatically generated

**Display menu choice**

Now this will display all the menu item that is received

A screenshot of a computer

Description automatically generated

**Update order status choice**

With the order ID the food preparation will take 10 seconds to be cooked then it will be updated to ready to be served

A screenshot of a computer

Description automatically generated

Cancel order choice

This will cancel the order because maybe its out of stock

## The fifth file “ casher”

### Pseudo-code for cashier file

1. function log\_transaction(amount, payment\_method)

2. """Log each transaction to the transactions.txt file."""

3. try

4. open 'data/transactions.txt' as file

5. file.write f"{amount:.2f},{payment\_method}\n"

6. except Exception as e

7. print f"Error logging transaction: {e}"

8. end function

9. function log\_sale(item, quantity, amount)

10. """Log each item sold to the daily\_sales.txt file with correct quantity and total amount."""

11. try

12. open 'data/daily\_sales.txt' as file

13. file.write f"{item},{quantity},{amount \* quantity}\n" # Ensure amount reflects total sale

14. except Exception as e

15. print f"Error logging sale: {e}"

16. end function

17. function read\_order\_details(order\_id, takeaway)

18. """Fetch order details based on order ID."""

19. orders = []

20. total = 0

21. try

22. open 'data/orders.txt' as file

23. for each line in file

24. parts = split line by comma

25. # Ensure we're matching the order ID and expecting 7 parts per line

26. if parts[0] equals order\_id and length of parts equals 7

27. \_, \_, \_, \_, item, quantity, price = parts

28. quantity = convert quantity to integer

29. price = convert price to float

30. total\_price = quantity \* price

31. add "{item} x {quantity}: {total\_price:.2f}" to orders

32. total += total\_price

33. except FileNotFoundError

34. print "Orders file not found. Please ensure the 'data/orders.txt' file exists."

35. # Adjust for takeaway charge

36. if takeaway

37. total += length of orders # Assuming RM1 for each item if takeaway

38. return orders, total

39. end function

40. function generate\_cash\_bill()

41. """Generate a bill for an order."""

42. order\_id = input "Enter the order ID to generate bill: "

43. takeaway = input "Is this order takeaway? (yes/no): ".lower().startswith 'y'

44. orders, total = read\_order\_details(order\_id, takeaway)

45.

46. if not orders

47. print "Order not found."

48. return

49.

50. service\_charge = total \* 0.10

51. gst = total \* 0.06

52. final\_total = total + service\_charge + gst

53. if takeaway

54. takeaway\_charge = length of orders # RM1 for each item if takeaway

55. final\_total += takeaway\_charge

56. print "\n--- Bill Summary ---"

57. print "\n".join(orders)

58. print "Subtotal:", total

59. if takeaway

60. print "Takeaway Packing Charge:", takeaway\_charge

61. print "Service Charge (10%):", service\_charge

62. print "Government Service Tax (6%):", gst

63. print "Total:", final\_total

64.

65. payment\_method = input "Enter payment method (cash, e-wallet, debit, credit card): "

66. log\_transaction(final\_total, payment\_method)

67. # Log each sale

68. for each order\_line in orders

69. item, price = split order\_line by ': '

70. log\_sale item, 1, convert price to float # Log the sale with quantity as 1 for simplicity

71. print f"Payment of {final\_total:.2f} received via {payment\_method}. Thank you!"

72. end function

73. function end\_of\_day\_processing()

74. """Process transactions at the end of the day."""

75. print "End of Day Summary:"

76. total\_cash = total\_card = total\_ewallet = 0.0

77. try

78. open 'data/transactions.txt' as file

79. for each line in file

80. amount, method = split line by comma

81. amount = convert amount to float

82. if method equals "cash"

83. total\_cash += amount

84. elif method in ["debit", "credit card"]

85. total\_card += amount

86. elif method equals "e-wallet"

87. total\_ewallet += amount

88. except FileNotFoundError

89. print "Transactions file not found. No transactions for today."

90. return

91.

92. print "Total Cash Received:", total\_cash

93. print "Total Card Payments:", total\_card

94. print "Total E-Wallet Payments:", total\_ewallet

95. end function

96. function daily\_sales\_analytics()

97. sales\_data = {}

98. try

99. open 'data/daily\_sales.txt' as file

100. for each line in file

101. parts = split line by comma

102. if length of parts equals 3 # Ensure the line has three parts

103. item, quantity, amount = parts[0], parts[1], parts[2]

104. if item in sales\_data

105. sales\_data[item]['quantity'] += convert quantity to integer

106. sales\_data[item]['amount'] += convert amount to float

107. else

108. sales\_data[item] = {'quantity': convert quantity to integer, 'amount': convert amount to float}

109. else

110. print f"Skipping line due to unexpected format: {line.strip()}"

111. except FileNotFoundError

112. print "Daily sales file not found. Please ensure the 'data/daily\_sales.txt' file exists."

113. return

114.

115. if not sales\_data

116. print "No sales data available for today."

117. return

118.

119. total\_sales\_amount = sum item['amount'] for item in sales\_data.values()

120. total\_sales\_quantity = sum item['quantity'] for item in sales\_data.values()

121.

122. top\_selling\_item = max sales\_data.items() by key=lambda x: x[1]['quantity']

123. least\_selling\_item = min sales\_data.items() by key=lambda x: x[1]['quantity']

124.

125. print "\n--- Daily Sales Analytics ---"

126. print "Total Sales Amount:", total\_sales\_amount

127. print "Total Number of Sales:", total\_sales\_quantity

128. print "Top Selling Food Item:", top\_selling\_item[0], "(Quantity Sold:", top\_selling\_item[1]['quantity'], ")"

129. print "Least Selling Food Item:", least\_selling\_item[0], "(Quantity Sold:", least\_selling\_item[1]['quantity'], ")"

130. end function

131. function cashier\_interface()

132. while True

133. print "\n╔═══════════════════════════════════╗"

134. print "║ Cashier Interface ║"

135. print "╠═══════════════════════════════════╣"

136. print "║ 1. Generate Cash Bill ║"

137. print "║───────────────────────────────────║"

138. print "║ 2. End of Day Processing ║"

139. print "║───────────────────────────────────║"

140. print "║ 3. Daily Sales Analytics ║"

141. print "║───────────────────────────────────║"

142. print "║ 4. Exit ║"

143. print "╚═══════════════════════════════════╝"

144.

145. choice = input "Enter your choice (1-4): "

146.

147. if choice equals '1'

148. generate\_cash\_bill()

149. elif choice equals '2'

150. end\_of\_day\_processing() # Implement this based on your system's needs

151. elif choice equals '3'

152. daily\_sales\_analytics() # Implement this based on your system's needs

153. elif choice equals '4'

154. print "Exiting..."

155. return

156. else

157. print "Invalid choice, please try again."

158. end function

159. if \_\_name\_\_ equals "\_\_main\_\_"

160. cashier\_interface()

161. end if

### Cashier Service Source Code And Explanation

A screen shot of a computer program

Description automatically generated

**log\_transaction():**

The function does the log of every transaction to the file that is named transactions.txt Amount and payment\_method are two parameters that are being used for the transaction (method of payment).

**log\_sale():**

It retrieves the data such as the total amount, quantity from the file and then writes this info to the daily\_sales.txt file. It has only three parameters defined as item (the name of the item), quantity (quantity sold), and amount (the price per item).

**read\_order\_details():**

This function retrieves the order details from the orders.txt simply by providing the order ID as an argument. It placed an ordered list of items plus totalling of price by priority. It has two parts, order\_id( the ID of an order) and takeaway(a Boolean field for indicating whether the order is for takeaway or not).

A screen shot of a computer program

Description automatically generated

A screen shot of a computer code

Description automatically generated

**generate\_cash\_bill():**

With this function the bill is physicalized for an order. The script will ask for the order ID and whether the order was a takeaway or not. Following that, it works out the sub-total, service charge, GST, and the amount due. It marks the transaction activity and sales numbers too. In the end, it shows the bill summary page and gives the payment method as an option.

**end\_of\_day\_processing():**

This function reconciles transactions that took place at the end of the trading day. It takes the transactions from the transactions.txt file and put those into 3 different types of modes: cash, card and electronic wallet. Finally, it writes a report of amount cash payment, amount card payment and amount e-wallet payment.

A screen shot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

**daily\_sales\_analytics():**

This process reports on the daily sales data as contained in the daily\_sales.txt file. Through summing up the sales amount and number of sales, it shows the food item that sells the most and the one that sells the least. Finally, the summary statistics for the selected data points are given.

**cashier\_interface():**

This function is an evidence of an interface between the cashier and different tasks. It displays this menu with options for generation of a cash bill, conduct an end-of-day processing, view daily sales reports or an exit. Based on the usage of certain function, it calls the function accordingly.

### The Output of the code

**Generate cash bill choice**

This will ask the order id to generate the bill and it will also ask if the order is take away or no then he will choose how to pay then bill will be done successfully

A black screen with white text

Description automatically generated

This in the transactions.txt file



**End-of-day processing choice**

This will show all the different payments we got

A black rectangle with white text

Description automatically generated

**Daily sales analytics choice**

From the file, daily sales will print all details like what’s the most selling food today as shown in the picture below

A screenshot of a computer

Description automatically generated

**Exit choice** will just exit the system

# Conclusion:

The management system in a restaurant that is presented here is an all-inclusive solution which helps to simplify the restaurant's operations on various aspects such as the users' management, menu and table management, order process, and sales analysis. Based on a network of cooperative modules, the system facilitates seamless choreography of different functions complied by workers in a restaurant.

Key Features: User Management: The addition creates/edit/delete user accounts of different roles i.e. boss, cook, cooker and waiter/waitress. User authentication serves sole purpose of secure access to the system. Menu and Table Management: Managers can add, edit, or delete menu items, and also may assign an occupied, free or etc. status to any given table. This ensures that the restaurant in question is advertised in a manner which displays exact meals and room capacity.

Order Handling: Waiters/waitresses can keep an eye on customers, take orders from customers, and serve them without delay. Orders are directed to the kitchen for their preparing and after preparing they are served to the table for their respective customers. The software is designed to be able to monitor the status of order to provide prompt service.

Cashier Operations: Cashiers may be able to create bills, deal with transactions, as well as perform end-of-day processing. Sales analysis on a daily basis gives the restaurant the opportunities to understand the performance of the restaurant itself including the rate of total, sales, the most popular food, and the patterns of generated income.

Benefits: Efficiency: The system improves efficiency by the functions of automation and real-time updates, and it thus eliminates high rates of manual error. Customer Satisfaction: Improved order execution speed as well as better service both contribute to higher customer satisfaction and loyalty. Data-driven Decision Making: Sales analytics help company managers to make smart money decisions on such issues as menu adjustment, pricing and resource allocation. Future Enhancements: Integration of online request platforms for a smooth order processing. Customer relation management strategy that involves employing loyalty programs and customer feedback channels to further engage with customers. Predictive analytics for demand prediction and inventory management is one of the key features.

# Referencing

F. (2018, July 11). *Learn Python - Full Course for Beginners [Tutorial]*. YouTube. <https://www.youtube.com/watch?v=rfscVS0vtbw>

B. C. (2021, February 15). *Python Full Course for free 🐍*. YouTube. <https://www.youtube.com/watch?v=XKHEtdqhLK8>

P. W. M. (2018, November 6). *Python Functions | Python Tutorial for Absolute Beginners #1*. YouTube. <https://www.youtube.com/watch?v=u-OmVr_fT4s>

P. W. M. (2018, November 5). *Python For Loops - Python Tutorial for Absolute Beginners*. YouTube. <https://www.youtube.com/watch?v=94UHCEmprCY>

P. W. M. (2018, November 5). *Python For Loops - Python Tutorial for Absolute Beginners*. YouTube. <https://www.youtube.com/watch?v=94UHCEmprCY>