Table of Contents

Design of Program (Pseudocode) ………………………………………………… 5

Main Menu ………………………………………………………………… 5

Cinema Manager …………………………………………………………… 8

Technician ………………………………………………………………… 27

Customer …………………………………………………………………. 33

Ticketing Clerk …………………………………………………………… 46

Unique ID Numbers ……………………………………………………… 48

Function Explanation …………………………………………………………… 51

Text Files ………………………………………………………………………… 97

Runtime Operation ……………………………………………………………... 103

Assumptions Of System Developed ……………………………………………. 121

Advantages Of System Developed ……………………………………………... 123

List Of Figures

Figure 1 : Add Function …………………………………………………………. 51

Figure 2 : Update Function ……………………………………………………… 52

Figure 3 : Remove Function ……………………………………………………… 53

Figure 4 : Report Function (Reading) …………………………………………… 54

Figure 5 : Payment Methods (Report) Function ………………………………… 55

Figure 6 : Payment Methods (Report) (After cancellation) Function …………… 56

Figure 7 : Total Revenue (Report) Function ……………………………………… 57

Figure 8 : Total Revenue (Report) (After cancellation) Function ……………….. 58

Figure 9 : Total Seats Booked (Report) Function ………………………………… 59

Figure 10 : Total Seats Booked (Report) (After cancellation) Function ………… 60

Figure 11 : Payment Function …………………………………………………… 62

Figure 12 : Price Function ……………………………………………………….. 64

Figure 13 : Discounts Function ………………………………………………….. 65

Figure 14 : Food Function ……………………………………………………….. 68

Figure 15 : Food Collection Function …………………………………………… 70

Figure 16 : Price Printing Function ……………………………………………… 71

Figure 17 : Feedback Function …………………………………………………… 71

Figure 18 : Reading Feedback Function ………………………………………… 71

Figure 19 : Rewards Function …………………………………………………… 72

Figure 20 : Reading Rewards Function ………………………………………….. 73

Figure 21 : Penalty Function …………………………………………………….. 73

Figure 22 : Technical Issue (Reporting) Function ……………………………….. 74

Figure 23 : Reading Technical Issues ……………………………………………. 75

Figure 24 : Editing Technical Issues ……………………………………………... 77

Figure 25 : Load Users Function ………………………………………………… 78

Figure 26 : Login Function ………………………………………………………. 79

Figure 27 : New User Function …………………………………………………... 80

Figure 28 : Display Movie Function ……………………………………………... 81

Figure 29 : Booking Seats Function ……………………………………………… 83

Figure 30 : Seats Available Function …………………………………………….. 84

Figure 31 : Seats Available (Cancellation) Function …………………………….. 85

Figure 32 : Ticket Numbers Function ……………………………………………. 86

Figure 33 : Save Bookings Function ……………………………………………... 87

Figure 34 : Booking History Function …………………………………………… 88

Figure 35 : Book Tickets Function ………………………………………………. 89

Figure 36 : Cancel Booking Function …………………………………………… 90

Figure 37 : Receipt Function …………………………………………………….. 91

Figure 38 : Cash Payment Function ……………………………………………… 94

Figure 39 : Cinema ID Function (Unique ID Number) ………………………….. 95

Figure 40 : Payment ID Function (Unique ID Number) ………………………… 95

Figure 41 : Issue ID Function (Unique ID Number) …………………………….. 95

Figure 42 : Booking ID Function (Unique ID Number) ………………………… 96

Figure 43 : Food ID Function (Unique ID Number) …………………………….. 96

Figure 44 : Show Timings.txt …………………………………………………… 97

Figure 45 : Bookings.txt ………………………………………………………… 97

Figure 46 : Cash Payment.txt …………………………………………………… 98

Figure 47 : Food.txt ……………………………………………………………... 98

Figure 48 : Issues.txt …………………………………………………………….. 99

Figure 49 : Feedback.txt ……………………………………………………….... 99

Figure 50 : Overall Report.txt …………………………………………………... 100

Figure 51 : Seats.txt …………………………………………………………….. 100

Figure 52 : Payment.txt ………………………………………………………… 101

Figure 53 : Rewards.txt ………………………………………………………… 101

Figure 54 : Users.txt ……………………………………………………………. 102

Figure 55 : Register New User ………………………………………………… 103

Figure 56 : Login & Booking Sample 1 (Card Payment & Food Order) ……… 105

Figure 57 : Receipt File for Booking Sample 1 (Card Payment & Food Order) ... 105

Figure 58 : Login & Booking Sample 2 (Online Banking Payment) …………… 107

Figure 59 : Receipt File for Booking Sample 2 (Online Banking Payment) ……107

Figure 60 : Login & Booking Sample 3 (with Incorrect Password)

(E-Wallet Payment) …………………………………………………………….. 109

Figure 61 : Receipt File for Booking Sample 3 (with Incorrect Password)

(E-Wallet Payment) …………………………………………………………….. 109

Figure 62 : Login & Booking Sample 4 (Cash Payment) ……………………… 110

Figure 63 : View Booking History ……………………………………………… 111

Figure 64 : View Pricing ………………………………………………………... 111

Figure 65 : Reporting a Technical Issue ………………………………………… 112

Figure 66 : Giving Feedback …………………………………………………… 112

Figure 67 : Technician Reading Issues File ……………………………………... 113

Figure 68 : Technician Updating Issues File …………………………………… 113

Figure 69 : Ticketing Clerk (Cash Payment) …………………………………… 114

Figure 70 : Ticketing Clerk (In Store Food Order) ……………………………… 116

Figure 71 : Ticketing Clerk (Food Collection) …………………………………. 116

Figure 72 : Cinema Manager (Adding Movie Listing) ………………………… 117

Figure 73 : Cinema Manager (Updating Movie Listing) ………………………. 118

Figure 74 : Cinema Manager (Removing Movie Listing) ……………………… 119

Figure 75 : Cinema Manager (View Overall Report) …………………………… 120

Figure 76 : Cinema Manager (View Feedback) ………………………………… 120

Design Of Program (Pseudocode)

[MAIN MENU]

PRINT welcome message

DISPLAY main menu options:

1. Login

2. Register

3. View Pricing

4. Report Technical Issue

5. Feedback

PROMPT user for operation number → choice

IF choice == "1": // Login

IF login successful:

CALL rewards\_read(email)

DISPLAY post-login options:

1. Book Tickets

2. View Booking History

3. Cancel Booking

4. Logout

PROMPT user → c

IF c == "1":

CALL book\_tickets()

ELIF c == "2":

CALL booking\_hist()

ELIF c == "3":

CALL cancel\_booking()

ELIF c == "4":

PRINT logout message

ELSE:

PRINT invalid option

ELIF choice == "2": // Register

CALL new\_user()

ELIF choice == "3": // View Pricing

PRINT thank-you message

CALL Pricing()

ELIF choice == "4": // Report Technical Issue

PRINT login prompt

IF login successful:

CALL Technical\_Issue()

ELIF choice == "5": // Feedback

PRINT login prompt

IF login successful:

CALL Feedback()

ELIF choice.upper() == "ID": // Hidden backend portal

PROMPT for ID

IF ID == "Tech": // Technician

PRINT greeting

DISPLAY options:

1. View Technical Issues

2. Edit Technical Issues

PROMPT → op

IF op == "1":

CALL Technir()

ELIF op == "2":

CALL Techniw()

ELIF ID == "TicketCl": // Ticketing Clerk

PRINT greeting

DISPLAY options:

1. Customer Cash Payment

2. Booking Ticket for Customer

3. Food Order

4. Food Collection

PROMPT → proc

SET email = "In Store Booking"

IF proc == "1":

CALL CashP()

ELIF proc == "2":

CALL book\_tickets()

IF PM == "Cash":

CALL CashP()

ELIF proc == "3":

SET mode = "In Store"

CALL Food(mode)

SET penal = "No"

SET fo = "y"

CALL Pay(FoodP)

ELIF proc == "4":

CALL Food\_Collection()

ELIF ID == "CinemaMang": // Cinema Manager

PRINT greeting

DISPLAY options:

1. Add Movie Listings

2. Update Movie Listings

3. Remove Movie Listings

4. Overall Report

5. View Feedbacks

PROMPT → proc

IF proc == "1":

CALL AddC()

ELIF proc == "2":

CALL UpdateC()

ELIF proc == "3":

CALL RemoveC()

ELIF proc == "4":

CALL Overall\_Report()

ELIF proc == "5":

CALL Feedback\_read()

{CINEMA MANAGER}

[ADD MOVIE LISTINGS FUNCTION]

FUNCTION AddC

PROMPT user to enter Cinema Name

STORE the input in variable cname

FORMAT cname to Title Case

PROMPT user to enter Cinema Language

STORE the input in variable clang

FORMAT clang to Title Case

PROMPT user to enter Show Date (DD-MM-YYYY)

STORE the input in variable sdate

PROMPT user to enter Show Time

STORE the input in variable stime

PROMPT user to enter Cinema Location

STORE the input in variable clocation

FORMAT clocation to Title Case

PROMPT user to enter Hall Number

STORE the input in variable hallno

IF hallno is less than or equal to 4 THEN

SET seatcount to 120

ELSE

SET seatcount to 100

CALL Cinemaid function

STORE the result in variable showid

CREATE dictionary cn with the following key-value pairs:

"Cinema Name" = cname

"Cinema Language" = clang

"Show Date" = sdate

"Show Time" = stime

"Cinema Location" = clocation

"Hall Number" = hallno

"Seating Capacity" = seatcount

"Seats Available" = seatcount

"Show ID" = showid

CONVERT cn to string format

WRITE the string into STF file

END FUNCTION

[UPDATE MOVIE LISTINGS FUNCTION]

FUNCTION UpdateC():

OPEN STF file

READ all lines into list

PROMPT user for Show ID → shoid

INITIALIZE updated\_lines list

SET found = False

FOR each line in STF:

IF line contains Show ID:

PROMPT user for field to update → x

PROMPT user for new value → up

IF field x exists in line:

SPLIT line into key-value parts

FOR each part:

IF key matches x:

REPLACE value with up

ELSE:

KEEP original key-value

REBUILD line with updated field

ADD updated line to updated\_lines

PRINT "Data Updated Successfully"

SET found = True

ELSE:

PRINT "Field not found"

ADD original line to updated\_lines

CONTINUE to next line

ELSE:

ADD original line to updated\_lines

IF found == False:

PRINT "Specified Show ID does not exist"

OVERWRITE STF file with updated\_lines

FLUSH file

END FUNCTION

[REMOVE MOVIE LISTINGS FUNCTION]

FUNCTION RemoveC():

OPEN STF file

READ all lines into list

PROMPT user for Show ID → shoid

INITIALIZE new\_lines list

SET removed = 0

FOR each line in STF:

IF line contains Show ID:

PRINT "Movie Listing removed successfully"

INCREMENT removed

SKIP appending this line

ELSE:

STRIP line and append to new\_lines

IF removed == 0:

PRINT "Specified Show ID does not exist"

OVERWRITE STF file with new\_lines

FLUSH file

END FUNCTION

[REPORT FILE READING FUNCTION]

FUNCTION Overall\_Report

MOVE file pointer to start of ORF

READ all lines from ORF into variable lines

FOR each line in lines DO

PRINT the line

END FUNCTION

[REPORT (PAYMENT METHOD) FUNCTION]

FUNCTION OR\_Payment(x):

OPEN ORF file

READ all lines into list

SET updated = False

FOR each line in ORF:

STRIP line

IF line is a dictionary with key-value pairs:

PARSE line into payment dictionary

FOR each key-value pair:

CLEAN key

CONVERT value to integer (default to 0 if error)

ADD value to payment[key]

INCREMENT payment[x] by 1

REBUILD line from updated dictionary

REPLACE line in lines

SET updated = True

BREAK loop

IF no matching line was found:

APPEND new line with {x: 1}

OVERWRITE ORF file with updated lines

FLUSH file

END FUNCTION

[REPORT (PAYMENT METHOD) (CANCELLATION) FUNCTION]

FUNCTION cancel\_OR\_Payment(x):

OPEN ORF file

READ all lines into list

INITIALIZE payment dictionary

FOR each line in ORF:

IF line is a dictionary with key-value pairs:

PARSE line into key-value pairs

FOR each pair:

CLEAN key

CONVERT value to integer (default to 0 if error)

ADD value to payment[key]

IF x exists in payment AND payment[x] > 0:

DECREMENT payment[x] by 1

REBUILD dictionary line from updated payment

OVERWRITE ORF file with new line

FLUSH file

END FUNCTION

[REPORT (REVENUE) FUNCTION]

FUNCTION OR\_Revenue(a):

OPEN ORF file

READ all lines into list

FOR each line in ORF:

IF line starts with "Total Revenue =":

TRY to extract current revenue → cr

IF extraction fails:

SET cr = 0.0

ADD a to cr → n

FORMAT line as "Total Revenue = n"

REPLACE line in list

BREAK loop

IF no revenue line was found:

APPEND new line with "Total Revenue = a"

OVERWRITE ORF file with updated lines

FLUSH file

END FUNCTION

[REPORT (REVENUE) (CANCELLATION) FUNCTION]

FUNCTION cancel\_OR\_Revenue(a):

OPEN ORF file

READ all lines into list

OPEN PF file

READ all lines into list

SET refund\_amount = None

FOR each line in PF:

IF line contains Payment ID a:

SPLIT line into parts

FOR each part:

IF part contains 'Amount Paid':

TRY to extract value → refund\_amount

IF extraction fails:

SET refund\_amount = 0

BREAK loop

IF line contains 'Status':

REPLACE 'Paid' with 'Refunded.'

ELSE:

APPEND "'Status': 'Refunded.'" to line

UPDATE line in PF

BREAK loop

IF refund\_amount is not None:

CALL cancel\_OR\_Bookings(a)

FOR each line in ORF:

IF line starts with "Total Revenue =":

TRY to extract current revenue → current

IF extraction fails:

SET current = 0

SUBTRACT refund\_amount from current → new\_total

UPDATE line with new\_total

BREAK loop

PRINT refund confirmation message

OVERWRITE PF file with updated lines

FLUSH PF file

OVERWRITE ORF file with updated lines

FLUSH ORF file

END FUNCTION

[REPORT (TOTAL SEATS BOOKED) FUNCTION]

FUNCTION OR\_Bookings(a):

OPEN ORF file

READ all lines into list

FOR each line in ORF:

IF line starts with "Total Seats Booked =":

TRY to extract current seat count → cr

IF extraction fails:

SET cr = 0

ADD a to cr → n

FORMAT line as "Total Seats Booked = n"

REPLACE line in list

BREAK loop

IF no matching line was found:

APPEND new line with "Total Seats Booked = a"

OVERWRITE ORF file with updated lines

FLUSH file

END FUNCTION

[REPORT (TOTAL SEATS BOOKED) (CANCELLATION) FUNCTION]

FUNCTION cancel\_OR\_Bookings(a):

OPEN ORF file

READ all lines into list

OPEN PF file

READ all lines into list

SET seat\_count = 0

FOR each line in PF:

IF line contains Payment ID a:

SPLIT line into key-value parts

FOR each part:

IF part contains 'Number of Seats':

TRY to extract value → seat\_count

IF extraction fails:

SET seat\_count = 0

BREAK loop

BREAK loop

IF seat\_count > 0:

FOR each line in ORF:

IF line starts with "Total Seats Booked =":

TRY to extract current seat count → cr

IF extraction fails:

SET cr = 0

SUBTRACT seat\_count from cr → new\_total

UPDATE line with new\_total

BREAK loop

OVERWRITE ORF file with updated lines

FLUSH file

END FUNCTION

[PAYMENT FUNCTION]

FUNCTION Pay(amount):

PRINT empty line

IF penal == "Yes":

CALL Penalty()

DISPLAY "You have to Pay RM <amount>"

PROMPT user for payment method → store in PM

IF PM is one of ["Card", "Online Banking", "E-Wallet", "Cash"]:

INITIALIZE payment\_details dictionary

GENERATE unique PayID

IF PM == "Card":

SET attempt\_count = 0

WHILE attempt\_count < 3:

PROMPT for Card Number

IF invalid:

DISPLAY error

INCREMENT attempt\_count

ELSE:

PROMPT for Card Name, Validity, CVV

PROMPT for confirmation ("Pay Now")

IF confirmed:

DISPLAY success

STORE card details and amount in payment\_details

IF food ordered:

ADD Food ID to payment\_details

MARK status as "Paid"

WRITE payment\_details to Payment File

EXIT loop

ELSE:

DISPLAY failure

ELSE IF PM == "Online Banking":

PROMPT for Bank Name

DISPLAY redirection message

PROMPT for Username and Password

PROMPT for confirmation ("Pay Now")

IF confirmed:

DISPLAY success

STORE bank details and amount in payment\_details

IF food ordered:

ADD Food ID to payment\_details

MARK status as "Paid"

WRITE payment\_details to Payment File

ELSE:

DISPLAY failure

ELSE IF PM == "E-Wallet":

PROMPT for E-wallet Name

DISPLAY redirection message

PROMPT for confirmation ("Pay Now")

IF confirmed:

DISPLAY success

STORE e-wallet details and amount in payment\_details

IF food ordered:

ADD Food ID to payment\_details

MARK status as "Paid"

WRITE payment\_details to Payment File

ELSE:

DISPLAY failure

ELSE IF PM == "Cash":

GENERATE unique CashID

DISPLAY walk-in instructions

DISPLAY CashID for counter payment

INITIALIZE to\_be\_paid dictionary with booking details

IF food ordered:

ADD Food ID to to\_be\_paid

MARK status as "Not Paid"

WRITE to\_be\_paid to Cash Payment File

CALL OR\_Payment(PM) to update payment method count

ELSE:

DISPLAY "Enter a Valid Payment Method."

END FUNCTION

[CALCULATING PRICE FUNCTION]

FUNCTION Price(x, y, z)

SET Adult Ticket Price to 20

STORE value into AdultP

SET Child Ticket Price to 15

STORE value into ChildP

SET Senior Citizen / OKU Ticket Price to 15

STORE value into SaOP

CALCULATE (x × AdultP) + (y × ChildP) + (z × SaOP)

STORE result into TotalP

SET GST rate to 6%

STORE value into GST

CALCULATE TotalP × GST

STORE result into GSTP

ADD GSTP to TotalP

STORE result back into TotalP

DISPLAY "Total Price to Pay : RM", TotalP

END FUNCTION

[DISCOUNTS FUNCTION]

FUNCTION Discounts

SET counter to 0

STORE value into c

WHILE c is less than 3 DO

PROMPT user to enter Discount Code

STORE input into Dis

FORMAT Dis into Upper Case

IF Dis equals "H" THEN

CALCULATE TotalP × 30%

STORE result into DisP

EXIT loop

ELSE IF Dis equals "M" THEN

CALCULATE TotalP × 20%

STORE result into DisP

EXIT loop

ELSE IF Dis equals "L" THEN

CALCULATE TotalP × 10%

STORE result into DisP

EXIT loop

ELSE

DISPLAY "Enter a valid discount code."

INCREMENT c by 1

IF no valid code entered after 3 attempts THEN

DISPLAY "No Valid Code Entered."

RETURN from function

SUBTRACT DisP from TotalP

STORE result back into TotalP

DISPLAY "Total Price After Discount :", TotalP

END FUNCTION

[FOOD BOOKING FUNCTION]

FUNCTION Food(mode):

INITIALIZE food dictionary

GENERATE unique foodid

SET FoodP = 0

PROMPT user: "How many items do you want to order?" → c

IF c <= 0:

DISPLAY "Please enter a positive number."

RETURN

IF mode == "Online" AND c > number of seats booked:

DISPLAY "You cannot order more items than seats booked."

RETURN

SET count = 1

WHILE count <= c:

DISPLAY food menu:

1. Popcorn Box

2. Combo Box (Popcorn + Drink)

3. Drinks Only

PROMPT user for item number → f

IF f not in {"1", "2", "3"}:

DISPLAY "Invalid item number."

RETURN

IF f == "1": // Popcorn Only

PROMPT for flavour → spf

IF spf not in valid flavours:

DISPLAY "Invalid flavour."

RETURN

PROMPT for size → RoL

IF RoL == "Regular":

ADD item to food dictionary

INCREMENT FoodP by 13

ELIF RoL == "Large":

ADD item to food dictionary

INCREMENT FoodP by 15

ELSE:

DISPLAY "Invalid size."

ELIF f == "2": // Combo

PROMPT for flavour → spf

IF spf not in valid flavours:

DISPLAY "Invalid flavour."

RETURN

PROMPT for popcorn size → RoL

IF RoL == "Regular":

ADD item to food dictionary

INCREMENT FoodP by 13

ELIF RoL == "Large":

ADD item to food dictionary

INCREMENT FoodP by 15

ELSE:

DISPLAY "Invalid size."

DISPLAY drink menu:

1. Soft Drinks

2. Hot/Iced Drinks

3. Water Bottle

PROMPT for drink category → Drink

IF Drink == "1":

PROMPT for drink name → D

PROMPT for size → RoL

IF RoL == "Regular":

ADD drink to food dictionary

INCREMENT FoodP by 6

ELIF RoL == "Large":

ADD drink to food dictionary

INCREMENT FoodP by 7

ELSE:

DISPLAY "Invalid size."

ELIF Drink == "2":

PROMPT for drink name → D

PROMPT for Hot/Iced → HoC

IF HoC in {"Hot", "Iced"}:

ADD drink to food dictionary

INCREMENT FoodP by 7

ELSE:

DISPLAY "Invalid temperature."

ELIF Drink == "3":

PROMPT for size → RoL

IF RoL == "Regular":

ADD drink to food dictionary

INCREMENT FoodP by 3

ELIF RoL == "Large":

ADD drink to food dictionary

INCREMENT FoodP by 4

ELSE:

DISPLAY "Invalid size."

ELSE:

DISPLAY "Invalid drink selection."

RETURN

ELIF f == "3": // Drink Only

DISPLAY drink menu

PROMPT for drink category → Drink

IF Drink == "1":

PROMPT for drink name → D

PROMPT for size → RoL

IF RoL == "Regular":

ADD drink to food dictionary

INCREMENT FoodP by 6

ELIF RoL == "Large":

ADD drink to food dictionary

INCREMENT FoodP by 7

ELSE:

DISPLAY "Invalid size."

ELIF Drink == "2":

PROMPT for drink name → D

PROMPT for Hot/Iced → HoC

IF HoC in {"Hot", "Iced"}:

ADD drink to food dictionary

INCREMENT FoodP by 7

ELSE:

DISPLAY "Invalid temperature."

ELIF Drink == "3":

PROMPT for size → RoL

IF RoL == "Regular":

ADD drink to food dictionary

INCREMENT FoodP by 3

ELIF RoL == "Large":

ADD drink to food dictionary

INCREMENT FoodP by 4

ELSE:

DISPLAY "Invalid size."

ELSE:

DISPLAY "Invalid drink selection."

RETURN

INCREMENT count

IF food dictionary is empty:

DISPLAY "No valid food selected."

RETURN

IF mode == "Online":

ADD FoodP to TotalP

ELIF mode == "In Store":

DISPLAY "Total Amount: <FoodP>"

MARK food["Status"] = "Collected"

ADD current timestamp to food["Collected At"]

CALL OR\_Revenue(FoodP)

ADD email and foodid to food dictionary

WRITE food dictionary to FF file

END FUNCTION

[FOOD COLLECTION FUNCTION]

FUNCTION Food\_Collection():

OPEN FF file

READ all lines into list

PROMPT user for Food ID → fooid

INITIALIZE updated\_lines list

SET collected = False

FOR each line in FF:

IF line contains Food ID:

DISPLAY line to user

IF line contains 'Status':

REPLACE 'Paid' or 'Not Paid' with 'Collected'

ELSE:

APPEND "'Status': 'Collected'" to line

IF line contains 'Collected At':

SPLIT line at 'Collected At'

REPLACE timestamp with current datetime

ELSE:

APPEND "'Collected At': <current datetime>" to line

SET collected = True

ADD updated line to updated\_lines

ELSE:

ADD original line to updated\_lines

IF collected == False:

DISPLAY "No Food Orders Found."

OVERWRITE FF file with updated\_lines

FLUSH file

END FUNCTION

[PRINTING PRICING TO USER FUNCTION]

FUNCTION Pricing

STORE the following string into CPT:

"Cinema Ticket Pricing :

Adult (12years and Above) = RM20

Children (12years and Below) = RM15

Senior Citizens/OKU = RM15"

STORE the following string into FPT:

"Food Items Pricing :

Popcorn (Regular/Large) = RM13 ~ RM15

Soft Drinks (Regular/Large) = RM6 ~ RM7

Milo/Tea/Coffee (Hot/Iced) = RM7"

DISPLAY CPT

DISPLAY FPT

END FUNCTION

[FEEDBACK FUNCTION]

FUNCTION Feedback

CREATE empty dictionary

STORE it into feedback

DISPLAY "We Value Your Feedback. Please Rate Your Booking and leave a comment."

PROMPT user to rate out of 5 stars

STORE input into r

PROMPT user to enter comment

STORE input into comment

STORE email into feedback["Feedback By"]

STORE r into feedback["Rating"]

STORE comment into feedback["Comments"]

WRITE feedback as string into FBF

END FUNCTION

[FEEDBACK READING FUNCTION]

FUNCTION Feedback\_read

MOVE file pointer to beginning of FBF

READ entire content from FBF

STORE result into lines

DISPLAY lines

END FUNCTION

{TECHNICIAN]

[REWARDS FUNCTION]

FUNCTION rewards(em, r):

MOVE file pointer to start of RF

READ all lines into list → lines

INITIALIZE fr = 0

FOR each line in lines:

IF line contains "'Email': em":

IF line contains "'Rewards': 'No'":

INCREMENT fr

CREATE reward string:

'Email': em

'Rewards': r

IF r == "No":

IF fr + 1 > 5:

APPEND "'Penalty': 'Yes'" to reward

ELSE:

APPEND "'Penalty': 'No'" to reward

MOVE file pointer to start of RF

TRUNCATE RF

WRITE all original lines back (stripped + newline)

WRITE new reward entry as dictionary

FLUSH RF

END FUNCTION

[READING REWARDS FUNCTION]

FUNCTION rewards\_read(email)

MOVE file pointer to start of RF

READ all lines from RF into variable lines

SET global variable penal to "No"

FOR each line in lines DO

IF line contains "'Email': email" THEN

IF line contains "'Rewards': 'Yes'" THEN

DISPLAY discount message to user

ELSE IF line contains "'Penalty': 'Yes'" THEN

SET penal to "Yes"

DISPLAY penalty message to user

ELSE IF line contains "'Penalty': 'No'" THEN

SET penal to "No"

BREAK loop

END FUNCTION

[PENALTY FUNCTION]

FUNCTION Penalty

SET penalty amount to 50

STORE value into penalty

ADD penalty to TotalP

STORE result back into TotalP

DISPLAY "Your Penalty Has Been Added."

DISPLAY "Total Amount : RM", TotalP

END FUNCTION

[REPORTING TECHNICAL ISSUE FUNCTION]

FUNCTION Technical\_Issue

CREATE empty dictionary

STORE it into TP

INITIALIZE count to 0

WHILE count < 3 DO

PROMPT user to describe the technical issue

STORE input into ti

IF ti is empty or only whitespace THEN

DISPLAY "Technical Issue cannot be empty."

INCREMENT count by 1

ELSE

EXIT loop

IF count reaches 3 THEN

DISPLAY "Too Many Failed Attempts."

RETURN from function

INITIALIZE c to 0

WHILE c < 3 DO

PROMPT user to enter Cinema ID (from receipt)

STORE input into cid

IF cid is numeric THEN

EXIT loop

ELSE

DISPLAY "Cinema ID Must Be A Number."

INCREMENT c by 1

IF c reaches 3 THEN

DISPLAY "Too Many Failed Attempts."

RETURN from function

PROMPT user to enter time when issue started

STORE input into toi

CALL IssueID()

STORE result into issueid

STORE email into TP["Reported by"]

STORE ti into TP["Technical Issue"]

STORE toi into TP["Time of Issue"]

STORE cid into TP["Cinema ID"]

STORE issueid into TP["Issue ID"]

STORE "Not Fixed" into TP["Current"]

WRITE TP as string into TF

END FUNCTION

[READING TECHNICAL ISSUES FILE FUNCTION]

FUNCTION Technir

MOVE file pointer to start of TF

READ all lines from TF into variable lines

FOR each line in lines DO

STRIP whitespace from line

DISPLAY the line

END FUNCTION

[UPDATING TECHNICAL ISSUES FILE FUNCTION]

FUNCTION Techniw

CALL Technir to display current technical issues

MOVE file pointer to start of TF

READ all lines from TF into variable lines

PROMPT user to enter Issue ID

STORE input in variable iid

PROMPT user to choose operation:

1. Update an Issue

2. Issue Fixed

3. No Issue

STORE input in variable p

GET current date and STORE in variable ts

INITIALIZE updated\_lines as empty list

FOR each line in lines DO

IF line contains "'Issue ID': iid" THEN

INITIALIZE em as empty string

SPLIT line into parts by comma

FOR each part in parts DO

IF part contains "'Reported By':" THEN

EXTRACT email and STORE in em

BREAK loop

IF p equals "1" THEN

PROMPT user to enter field name to update

STORE input in variable x

IF field x not found in line THEN

DISPLAY "Invalid field name"

APPEND original line to updated\_lines

CONTINUE to next line

PROMPT user to enter updated value

STORE input in variable up

INITIALIZE new\_parts as empty list

FOR each part in parts DO

SPLIT part into key and value

IF key matches x THEN

APPEND updated key-value pair to new\_parts

ELSE

APPEND original part to new\_parts

APPEND "'Last Updated': ts" to new\_parts

JOIN new\_parts into new\_line

APPEND new\_line to updated\_lines

DISPLAY "Data Updated Successfully"

ELSE IF p equals "2" THEN

STRIP and prepare line for update

IF line contains "'Status':" THEN

REPLACE 'Pending' with 'Fixed'

ELSE

APPEND "'Status': 'Fixed'" to line

APPEND "'Last Updated': ts" to line

APPEND updated line to updated\_lines

CALL rewards(em, "Yes")

DISPLAY "Data Updated Successfully"

ELSE IF p equals "3" THEN

STRIP and prepare line for update

IF line contains "'Status':" THEN

REPLACE 'Pending' with 'No Issue'

ELSE

APPEND "'Status': 'No Issue'" to line

APPEND "'Last Updated': ts" to line

APPEND updated line to updated\_lines

CALL rewards(em, "No")

CONTINUE to next line

ELSE

APPEND original line to updated\_lines

IF lines are unchanged THEN

DISPLAY "Specified ID Doesn't Exist"

MOVE file pointer to start of TF

TRUNCATE TF

WRITE updated\_lines back to TF

FLUSH TF

END FUNCTION

{CUSTOMER}

[LOAD USERS FUNCTION]

FUNCTION load\_users

DECLARE global variable users

MOVE file pointer to start of UF

READ all lines from UF into variable lines

STRIP whitespace from each line

STORE cleaned lines in users list

END FUNCTION

[LOGIN FUNCTION]

FUNCTION login

CALL load\_users to populate users list

DECLARE global variable email

PROMPT user to enter email

STORE input in email (converted to lowercase)

IF email ends with "gmail.com" OR "yahoo.com" THEN

FOR each line in users DO

IF line contains "'Email': email" THEN

SPLIT line into parts by comma

INITIALIZE empty dictionary user\_data

FOR each part in parts DO

IF part contains ":" THEN

SPLIT part into key and value

STRIP quotes and whitespace

STORE key-value pair in user\_data

INITIALIZE count to 1

WHILE count is less than or equal to 3 DO

PROMPT user to enter password

IF password is empty THEN

DISPLAY "Password cannot be empty"

CONTINUE loop

ELSE IF password length is less than 6 THEN

DISPLAY "Password must be at least 6 characters"

CONTINUE loop

ELSE IF password matches user\_data["Password"] THEN

DISPLAY "Login Successful"

RETURN True

ELSE

DISPLAY "Password is wrong"

INCREMENT count

DISPLAY "Too many attempts. Please Try again later"

RETURN False

DISPLAY "No Profile Found with this Email"

PROMPT user to create new account (Y/N)

IF input is "y" THEN

CALL new\_user

ELSE IF input is "n" THEN

RETURN False

ELSE

DISPLAY "Session Terminated, Did not Enter A Valid Option"

RETURN False

ELSE

DISPLAY "Enter a valid Email ID"

RETURN False

END FUNCTION

[NEW USER FUNCTION]

FUNCTION new\_user

CALL load\_users to populate users list

PROMPT user to enter email

STORE input in user\_email (converted to lowercase)

IF user\_email ends with "gmail.com" OR "yahoo.com" THEN

FOR each line in users DO

IF line contains "'Email': user\_email" THEN

DISPLAY "Email Already Exists"

RETURN

INITIALIZE attempt counter i to 1

WHILE i is less than or equal to 3 DO

PROMPT user to enter password

PROMPT user to re-enter password

IF password length is less than 6 THEN

DISPLAY "Password must be at least 6 characters"

CONTINUE loop

IF password matches re-entered password THEN

FORMAT entry with email and password

MOVE file pointer to end of UF

WRITE entry to UF

FLUSH UF

DISPLAY "Account has been Created"

RETURN

ELSE

DISPLAY "Passwords aren't matching"

INCREMENT i

DISPLAY "Too many attempts. Please Try again later"

ELSE

DISPLAY "Enter a valid Email ID"

END FUNCTION

[DISPLAY MOVIE LISTING FUNCTION]

FUNCTION display\_movies():

PRINT empty line

MOVE file pointer to start of STF

READ all lines into list

PROMPT user: "Enter Movie Date (DD-MM-YYYY)" → date

FOR each line in STF:

IF line contains "'Show Date': date":

SPLIT line into key-value parts

FOR each part:

IF part contains "'Seats Available':":

TRY to extract seat count → seats

IF extraction fails:

SET seats = 0

IF seats > 0:

DISPLAY full line

BREAK inner loop

END FUNCTION

[BOOKING SEATS (SEAT NUMBERS) FUNCTION]

FUNCTION Booking\_seatlayout

DISPLAY blank line for spacing

INITIALIZE empty dictionary layouts

MOVE file pointer to start of STF

READ all lines from STF into variable lines

FOR each line in lines DO

IF line contains "'Show ID': mid" THEN

REMOVE braces and SPLIT line into parts by comma

INITIALIZE empty dictionary cn

FOR each part in parts DO

IF part contains ":" THEN

SPLIT part into key and value

STRIP quotes and whitespace

STORE key-value pair in cn

EXTRACT 'Cinema Name' into mn

EXTRACT 'Seating Capacity' into capacity (as integer)

TRY to open Seats.txt in read mode

FOR each line in Seats.txt DO

IF line contains ":" THEN

SPLIT line into fmd and layout

STORE layout as list in layouts[fmd]

EXCEPT FileNotFoundError THEN

PASS

IF mid not in layouts THEN

INITIALIZE layout with "O" repeated capacity times

DISPLAY seat layout header for mn

FOR each row\_num in layout with step size 10 DO

SLICE layout into row

COMPUTE row label (A, B, C, ...)

FORMAT row\_display with seat labels and status

DISPLAY formatted row

DISPLAY screen divider

INITIALIZE count to 0

WHILE count < 3 DO

DECLARE global variable seats

PROMPT user to enter seat numbers (e.g., A1,B3,C10)

SPLIT and clean input into seats list

IF seats list is empty THEN

DISPLAY error message

INCREMENT count

CONTINUE

IF seats contain duplicates THEN

DISPLAY error message

INCREMENT count

CONTINUE

IF number of seats equals nop THEN

SET valid to True

FOR each seat s in seats DO

VALIDATE format (letter + number)

COMPUTE row and column index

CHECK if index is within layout bounds

CHECK if seat is already booked

IF any check fails THEN

DISPLAY error

SET valid to False

BREAK

IF valid THEN

FOR each seat s in seats DO

COMPUTE index

MARK seat as booked ("X")

OPEN Seats.txt in write mode

FOR each fmn and layout in layouts DO

WRITE updated layout to file

BREAK loop

ELSE

INCREMENT count

ELSE IF number of seats < nop THEN

DISPLAY message to select more seats

INCREMENT count

ELSE IF number of seats > nop THEN

DISPLAY message to reduce seat count

INCREMENT count

RETURN

END FUNCTION

[SEATS AVAILABLE FUNCTION]

FUNCTION seats\_available(mid):

MOVE file pointer to start of STF

READ all lines into list → lines

INITIALIZE updated\_lines list

FOR each line in lines:

IF line contains Show ID mid:

SPLIT line into key-value parts

INITIALIZE movie dictionary

FOR each part:

IF part contains ":":

EXTRACT key and value

STRIP quotes and whitespace

ADD to movie dictionary

TRY to convert movie["Seats Available"] to integer → seats\_avail

IF conversion fails:

SET seats\_avail = 0

IF seats\_avail < number of seats requested (nop):

DISPLAY "Not Enough Seats Available."

ELSE:

SUBTRACT nop from seats\_avail

UPDATE movie["Seats Available"]

REBUILD line from updated movie dictionary

ADD new line to updated\_lines

CALL Booking\_seatlayout()

CONTINUE to next line

ELSE:

ADD original line to updated\_lines

OVERWRITE STF file with updated\_lines

FLUSH file

END FUNCTION

[SEAT AVAILABLE (CANCELLATION) FUNCTION]

FUNCTION cancel\_seats\_available(x, y)

MOVE file pointer to start of STF

READ all lines from STF into variable lines

INITIALIZE ul as empty list

FOR each line in lines DO

IF line contains "'Show ID': x" THEN

REMOVE braces and SPLIT line into parts by comma

INITIALIZE empty dictionary movie

FOR each part in parts DO

IF part contains ":" THEN

SPLIT part into key and value

STRIP quotes and whitespace

STORE key-value pair in movie

TRY to extract 'Seats Available' as integer

IF extraction fails THEN

SET seats\_avail to 0

INCREMENT seats\_avail by y

SET movie["Seats Available"] to max(0, seats\_avail)

FORMAT new\_line with updated movie dictionary

APPEND new\_line to ul

ELSE

APPEND original line to ul

MOVE file pointer to start of STF

TRUNCATE STF

WRITE updated ul back to STF

FLUSH STF

END FUNCTION

[TICKET NUMBERS FUNCTION]

FUNCTION ticket\_numbers

DISPLAY blank line for spacing

DECLARE global variables nop and mid

PROMPT user to enter Show ID

STORE input in mid

INITIALIZE count to 0

WHILE count < 3 DO

TRY to prompt user for:

Number of Adults (12 years and above)

Number of Children (12 years and below)

Number of Senior Citizens / OKU Members

IF any input is invalid THEN

DISPLAY "Please enter valid numbers"

INCREMENT count

CONTINUE loop

IF number of Adults <= 0 THEN

DISPLAY "There has to be at least 1 adult"

INCREMENT count

ELSE

BREAK loop

IF loop exited without valid input THEN

DISPLAY "Too many failed attempts. Please try again later"

RETURN

CALCULATE nop as sum of Adult + Child + SaO

CALL OR\_Bookings with nop

CALL seats\_available with mid

CALL Price with Adult, Child, SaO

END FUNCTION

[SAVE BOOKINGS FUNCTION]

FUNCTION save\_bookings():

MOVE file pointer to start of STF

READ all lines into list

GENERATE current timestamp → timestamp

GENERATE unique Booking ID → bookid

FOR each line in STF:

IF line contains Show ID mid:

SPLIT line into key-value parts

INITIALIZE cn dictionary

FOR each part:

IF part contains ":":

EXTRACT key and value

STRIP quotes and whitespace

ADD to cn dictionary

CREATE entry dictionary with:

"Booked By" = email

"Movie Name" = cn["Cinema Name"]

"Hall No." = cn["Hall Number"]

"Cinema Date" = cn["Show Date"]

"Location" = cn["Cinema Location"]

"No. of Tickets" = nop

"Seats" = seats

"Date & Time of Booking" = timestamp

"Show ID" = mid

"Payment ID" = Payid

"Booking ID" = bookid

IF food was ordered (fo == "y"):

ADD "Food" = foodid to entry

FORMAT entry as dictionary string

WRITE entry to BF file

FLUSH BF file

BREAK loop

END FUNCTION

[BOOKING HISTORY FUNCTION]

FUNCTION booking\_hist():

DISPLAY "Booking History"

MOVE file pointer to start of BF

READ all lines into list

INITIALIZE records = 0

FOR each line in BF:

IF line contains user's email:

DISPLAY line

INCREMENT records

IF records == 0:

DISPLAY "No Booking History Found."

ELSE:

DISPLAY "Records Found = <records>"

END FUNCTION

[BOOK TICKETS FUNCTION]

FUNCTION book\_tickets

CALL display\_movies to show available shows

CALL ticket\_numbers to collect ticket counts and validate

PROMPT user: "Do You Have Discount Codes?"

IF user input is "Y" THEN

CALL Discounts to apply discount

DECLARE global variable fo

PROMPT user: "Do You Want to Order Food? (Y/N)"

STORE input in fo

IF fo equals "y" THEN

SET mode to "Online"

CALL Food with mode

CALL Pay with TotalP

IF PM equals "Cash" THEN

RETURN (no further processing)

ELSE

CALL OR\_Revenue with TotalP

CALL save\_bookings to store booking record

CALL receipt to generate and display receipt

END FUNCTION

[CANCEL BOOKING FUNCTION]

FUNCTION cancel\_booking

MOVE file pointer to start of BF

READ all lines from BF into variable lines

INITIALIZE ul as empty list

FOR each line in lines DO

IF line contains "'Booked By': email" OR "'Email': email" THEN

DISPLAY the line

PROMPT user to enter Booking ID

STORE input in bid

IF line contains "'Booking ID': bid" THEN

IF line contains "'Status': 'Cancelled'" THEN

DISPLAY "This booking has already been cancelled"

RETURN

DISPLAY "Cancelling Booking..."

REMOVE braces and SPLIT line into parts by comma

INITIALIZE empty dictionary entry

FOR each part in parts DO

IF part contains ":" THEN

SPLIT part into key and value

STRIP quotes and whitespace

STORE key-value pair in entry

EXTRACT 'Show ID' into shid

TRY to extract 'No. of Tickets' into nt (as integer)

IF extraction fails THEN

SET nt to 0

CALL cancel\_seats\_available with shid and nt

CALL cancel\_OR\_Payment with entry["Payment ID"]

CALL cancel\_OR\_Revenue with entry["Payment ID"]

SET entry["Status"] to "Cancelled"

FORMAT new\_line from updated entry dictionary

APPEND new\_line to ul

DISPLAY "Booking Has Been Cancelled"

CONTINUE to next line

ELSE

APPEND original line to ul

MOVE file pointer to start of BF

TRUNCATE BF

WRITE updated ul back to BF

FLUSH BF

END FUNCTION

{TICKETING CLERK}

[RECEIPT FUNCTION]

FUNCTION receipt

DISPLAY blank line for spacing

MOVE file pointer to start of STF

READ all lines from STF into variable lines

DECLARE global dictionary cn

FOR each line in lines DO

IF line contains "'Show ID': mid" THEN

REMOVE braces and SPLIT line into parts by comma

INITIALIZE empty dictionary cn

FOR each part in parts DO

IF part contains ":" THEN

SPLIT part into key and value

STRIP quotes and whitespace

STORE key-value pair in cn

FORMAT receipt\_text with:

- Date & Time of Booking

- Customer Email

- Movie Name

- Movie Date

- Hall Number

- Location

- Number of Tickets

- Seat Labels

- Total Amount

- Booking ID

- Payment Method

IF fo equals "y" THEN

ADD Food ID to receipt\_text

APPEND thank-you message to receipt\_text

DISPLAY receipt\_text

GENERATE filename using email and date

OPEN file with filename in write mode

WRITE receipt\_text to file

DISPLAY confirmation message with filename

BREAK loop

END FUNCTION

[CASH PAYMENT FUNCTION]

FUNCTION CashP():

MOVE file pointer to start of CPF

READ all lines into list → lines

INITIALIZE updated\_lines list

PROMPT user: "Enter Payment ID" → Payid

FOR each line in lines:

IF line contains Payment ID:

DISPLAY line

PARSE line manually:

STRIP outer braces

SPLIT by commas, respecting nested lists

BUILD entry dictionary from key-value pairs

IF key == "Seats":

CONVERT string list to actual list

PROMPT user: "Paid (Y/N)" → pay

IF pay == "y":

SET entry["Status"] = "Paid"

EXTRACT global variables:

mid, nop, seats, fo, foodid, TotalP, PM, email

LOOKUP Show Date from STF using mid → date

CALL OR\_Revenue(TotalP)

CALL OR\_Bookings(nop)

CALL save\_bookings()

CALL receipt()

REPLACE 'Not Paid' with 'Paid' in line

ADD updated line to updated\_lines

ELIF pay == "n":

DISPLAY "Booking Cancelled"

SET entry["Status"] = "Cancelled"

REPLACE 'Not Paid' with 'Cancelled' in line

ADD updated line to updated\_lines

ELSE:

ADD original line to updated\_lines

ELSE:

ADD original line to updated\_lines

OVERWRITE CPF file with updated\_lines

FLUSH file

END FUNCTION

{FOR UNIQUE ID NUMBERS}

[CINEMA ID FUNCTION]

DECLARE ciid as global set

FUNCTION Cinemaid

LOOP indefinitely DO

GENERATE random number between 00000 and 99999

STORE result into num

IF num not in ciid THEN

ADD num into ciid

RETURN num

END FUNCTION

[PAY ID FUNCTION]

DECLARE pid as global set

FUNCTION PayID

LOOP indefinitely DO

GENERATE random number between 00000 and 99999

STORE result into payid

IF payid not in pid THEN

ADD payid into pid

RETURN payid

END FUNCTION

[ISSUE ID FUNCTION]

DECLARE iid as global set

FUNCTION IssueID

LOOP indefinitely DO

GENERATE random number between 00000 and 99999

STORE result into issueid

IF issueid not in iid THEN

ADD issueid into iid

RETURN issueid

END FUNCTION

[BOOKING ID FUNCTION]

DECLARE bid as global set

FUNCTION BookingID

LOOP indefinitely DO

GENERATE random number between 00000 and 99999

STORE result into bookingid

IF bookingid not in bid THEN

ADD bookingid into bid

RETURN bookingid

END FUNCTION

[FOOD ID FUNCTION]

DECLARE fid as global set

FUNCTION FoodID

LOOP indefinitely DO

GENERATE random number between 00000 and 99999

STORE result into foodid

IF foodid not in fid THEN

ADD foodid into fid

RETURN foodid

END FUNCTION

Function Explanation

Add Movie Listings Function

A computer code on a black background

AI-generated content may be incorrect.

*Figure 1 : Add Function*

This Function is used to Add Movie Listings into ‘Show Timings’ text file (STF opens ‘Show Timings.txt’ in a+ mode) which later is read and printed the data to the customer to select a movie to book tickets.

Important data like Cinema Name, Language, Date, Location, Hall Number, Seat capacity & Seats available are taken as input from the Cinema Manager.

Cinema Name and Language are stored in title format.

Based on the Hall Number, the Seating capacity and Seats available are updated.

All the data are saved in a dictionary format and written into the text file as a string (text files only accept strings).

Update Movie Listings Function

A screen shot of a computer screen

AI-generated content may be incorrect.

*Figure 2 : Update Function*

This Function is used to Update Movie Listings in ‘Show Timings’ text file (STF opens ‘Show Timings.txt’ in a+ mode).

STF.seek(0) resets the cursor position to 0 (starting of the file) so it reads the whole file.

STF.readlines() reads all the lines of the text file as a List of Strings.

[ast.literal\_eval(.strip()) for line in lines] is to convert the string (read from the text file) back into the python dictionary.

An input is asked to Cinema Manager to enter the ‘Show ID’ to update the particular movie listing. Then it is checked if the ID entered matches any of the movie listings in the file.

When the ID matches, the Cinema Manager enters the key to the value to be changed and then the updated value which is then updated into the dictionary.

If no movie listing is found with the ID entered, it prints “Specified Show ID doesn’t Exist.”

The file is then rewritten with the updated data.

Remove Movie Listing Function

A computer screen shot of a program code

AI-generated content may be incorrect.

*Figure 3 : Remove Function*

This Function is used to Remove Movie Listings in ‘Show Timings’ text file (STF opens ‘Show Timings.txt’ in a+ mode).

STF.seek(0) resets the cursor position to 0 (starting of the file) so it reads the whole file.

STF.readlines() reads all the lines of the text file as a List of Strings.

[ast.literal\_eval(.strip()) for line in lines] is to convert the string (read from the text file) back into the python dictionary.

An input is asked to Cinema Manager to enter the ‘Show ID’ to update the movie listing. Then it is checked if the ID entered matches any of the movie listings in the file.

When the ID matches, the movie listing is removed from the file.

If no movie listing is found with the ID entered, it prints “Specified Show ID doesn’t Exist.”

The file is then rewritten with the updated data.

Report Function (Reading File)

A black rectangle with white text

AI-generated content may be incorrect.

*Figure 4 : Report Function (Reading)*

This Function is used to read Overall Report data in ‘Overall Report’ text file (ORF opens ‘Overall Report.txt’ in a+ mode).

ORF.seek(0) resets the cursor position to 0 (starting of the file) so it reads the whole file.

ORF.readlines() reads all the lines of the text file as a List of Strings.

Prints the data line by line.

Payment Methods Used Function (Helper for Report)

A computer screen shot of code

AI-generated content may be incorrect.

*Figure 5 : Payment Methods (Report) Function*

This Function is used to read Overall Report data in ‘Overall Report’ text file (ORF opens ‘Overall Report.txt’ in a+ mode).

ORF.seek(0) resets the cursor position to 0 (starting of the file) so it reads the whole file.

ORF.readlines() reads all the lines of the text file as a List of Strings and then finds for the data that’s in dictionary format.

Then it updates the count for the payment method used (Eg: If Card method is used the count for ‘Card’ increases by 1. If it’s not found it creates a new entry).

It deletes all the old data and rewrites it with the new updated data.

Payment Methods Used Function (Helper for Report) (After Cancellation)

A computer screen shot of a code

AI-generated content may be incorrect.

*Figure 6 : Payment Methods (Report) (After cancellation) Function*

This Function is used to read Overall Report data in ‘Overall Report’ text file (ORF opens ‘Overall Report.txt’ in a+ mode).

ORF.seek(0) resets the cursor position to 0 (starting of the file) so it reads the whole file.

ORF.readlines() reads all the lines of the text file as a List of Strings and then finds for the data that’s in dictionary format.

Then it updates the count for the payment method used (Eg: If Card method is used the count for ‘Card’ decreases by 1).

It deletes all the old data and rewrites it with the new updated data.

Total Revenue Function (Helper for Report)

A computer code on a black background

AI-generated content may be incorrect.

*Figure 7 : Total Revenue (Report) Function*

This Function is used to read Overall Report data in ‘Overall Report’ text file (ORF opens ‘Overall Report.txt’ in a+ mode).

ORF.seek(0) resets the cursor position to 0 (starting of the file) so it reads the whole file.

ORF.readlines() reads all the lines of the text file as a List of Strings and then finds for the data that starts with “Total Revenue = ”.

Then it updates the total revenue (Adds all the money collected with ticket and food sales).

It deletes all the old data and rewrites it with the new updated data.

Total Revenue Function (Helper for Report) (After Cancellation)

A screen shot of a computer program

AI-generated content may be incorrect.

*Figure 8 : Total Revenue (Report) (After cancellation) Function*

This Function is used to read Overall Report data in ‘Overall Report’ text file (ORF opens ‘Overall Report.txt’ in a+ mode).

ORF.seek(0) resets the cursor position to 0 (starting of the file) so it reads the whole file.

ORF.readlines() reads all the lines of the text file as a List of Strings and then finds for the data that starts with “Total Revenue = ”.

Then it updates the total revenue (Subtracts the money for ticket and food paid by the customer (for that ticket)).

It deletes all the old data and rewrites it with the new updated data.

Total Seats Booked Function (Helper for Report)

A computer screen shot of text

AI-generated content may be incorrect.

*Figure 9 : Total Seats Booked (Report) Function*

This Function is used to read Overall Report data in ‘Overall Report’ text file (ORF opens ‘Overall Report.txt’ in a+ mode).

ORF.seek(0) resets the cursor position to 0 (starting of the file) so it reads the whole file.

ORF.readlines() reads all the lines of the text file as a List of Strings and then finds for the data that starts with “Total Seats Booked = ”.

Then it updates the total revenue (Adds the number of seats booked).

It deletes all the old data and rewrites it with the new updated data.

Total Seats Booked Function (After cancellation)(Helper for Report)

A screen shot of a computer

AI-generated content may be incorrect.

*Figure 10 : Total Seats Booked (Report) (After cancellation) Function*

This Function is used to read Overall Report data in ‘Overall Report’ text file (ORF opens ‘Overall Report.txt’ in a+ mode).

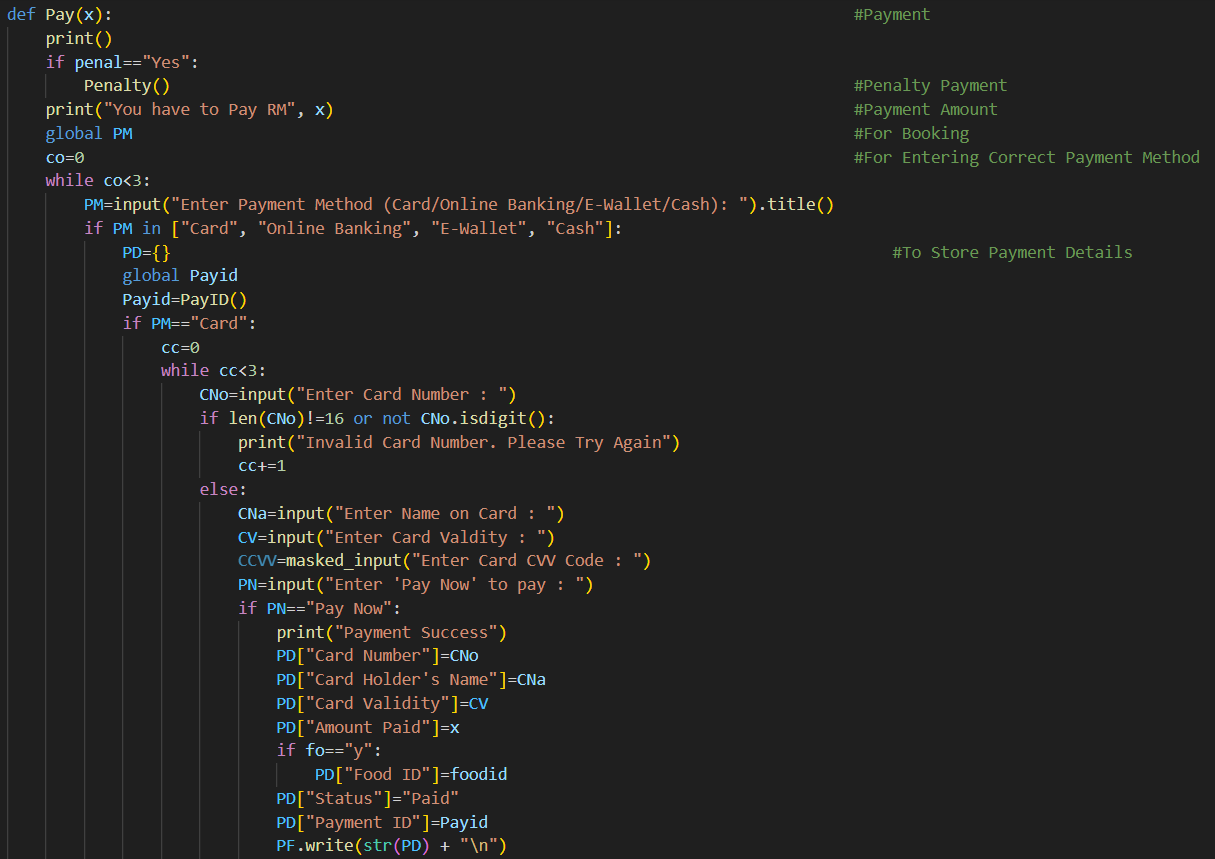
ORF.seek(0) resets the cursor position to 0 (starting of the file) so it reads the whole file.

ORF.readlines() reads all the lines of the text file as a List of Strings and then finds for the data that starts with “Total Seats Booked = ”.

Then it updates the total revenue (Subtracts the number of seats booked by the customer (for that ticket)).

It deletes all the old data and rewrites it with the new updated data.

Payment Function



A computer screen shot of a program

AI-generated content may be incorrect.

A screen shot of a computer screen

AI-generated content may be incorrect.

*Figure 11 : Payment Function*

This Function opens 2 text files, ‘Payment.txt’ (opened as PF) and ‘Cash Payment.txt’ (opened as CPF) which are opened in ‘a+’ mode.

Customer gets 4 different payment method options in which ‘Card’, ‘Online Banking’ & ‘E-Wallet’ payments get written into ‘Payments.txt’ while ‘Cash’ payment method is written into ‘Cash Payments.txt’ for easier callback for payment in-store.

For Card method, it asks for the Card number and checks if the length of the number entered is equal to 16 or not (checking if its valid number). If it’s valid, asks for Name on Card, Card Expiry Date, CVV code (taken in hidden input). All the data entered (except CVV code) is stored in ‘payment’ dictionary with food id (if purchased), payment id & amount paid and written into the text file.

For Online Banking method, it asks for Bank name and then prints “Redirecting to Online Banking…” and asks for bank’s user id and password (for bank login) and asks for a payment verification by asking customer to enter ‘Pay Now’. Then it stores the data like Bank name, Amount paid, food id (if purchased), payment id & status as ‘Paid’ and writes into the text file.

For E-Wallet Method, it asks for E-Wallet Service name (like Grab, TNG) and prints “Redirecting to E-Wallet site…” and asks for a payment verification by asking customer to enter ‘Pay Now’. Then it stores data like E-Wallet service name, Amount paid, food id (if purchased), payment id & status as ‘Paid’ and writes into the text file.

For Cash Payment, it gives a 5-digit unique id to the customer to use to complete the payment (in store) and then get the Receipt. It stores data like Amount to be paid, Booked by, Seat numbers, food id (if purchased), payment id (the one used to complete the payment) & status as ‘Not Paid’ and writes into the Cash Payments text file.

Then it calls the OR\_Payments Function (For Payment Methods used count) to update the count.

If a non-valid method is entered it prints “Enter a Valid Payment Method” and loops again (limited to 3 tries).

Price Function (Calculating Amount to Pay)

A black rectangle with white text

AI-generated content may be incorrect.

*Figure 12 : Price Function*

This Function is used to calculate the amount to be paid by the customer (only for cinema tickets).

The price varies depending on the age, 20 RM per Adult , 15 RM per Child and a discounted price of 15 RM for Senior Citizens and OKU members.

The number of tickets is taken as input in another function and this function is called.

After Calculation, GST (6%) for the tickets is calculated and added to the price.

Finally prints “Total Price to Pay : RM”,TotalP to the customer where ‘TotalP’ is the amount to pay.

Discounts Function

A computer screen with text on it

AI-generated content may be incorrect.

*Figure 13 : Discounts Function*

This Function is used to calculate the discounted price and print it to the user.

It asks the user if they have a Discount Code, if yes then this function is called or else it is skipped (This occurs in the Booking Function).

3 Different Discount codes are available, the customer is expected to enter the discount code and it is checked if it matches with the available discount codes, if yes, the discount is applied and the discounted price is printed to the customer.

Food Function

A screen shot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

A computer screen shot of text

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

A computer screen with text on it

AI-generated content may be incorrect.

*Figure 14 : Food Function*

This Function is used to order Food (Online and In Store) and writes into ‘Food.txt’ text file (FF opens ‘Food.txt’ in a+ mode).

Global foodid (to add in the receipt), FoodP (For payment of food price for in store buying).

Then Asks the user to input ‘number of food items’ and if online booking, it checks if the number of items is more than the number of seat tickets (limited to the number of tickets).

There are 3 different options, Only Popcorn box, Combo Pack (1 Popcorn Box & 1 Drink), and only Drink option.

For Popcorn there are 3 different flavors (Salted, Caramel & Mixed) and 2 different box sizes (Regular & Large).

For Drinks there are 3 main options, Soft Drinks (Coke, Sprite, Pepsi, A&W, Minute Maid Orange), Hot/Cold Drinks (Milo, Tea, Coffee) and Water Bottle. The Soft Drinks and Water Bottle are of 2 different sizes (Regular & Large).

Combo Pack has the same options but comes together.

If booked online with cinema ticket the Food Price is added to the Total Price and asked to pay.

If booked in store ‘Status’ is updated to “Collected” and ‘Collected Time’ noting the time of the collection of the food. OR\_Revenue Function is passed for this to add the money received into Total Revenue.

Food dictionary containing data like ‘Ordered By’, ‘Food ID’, Food ordered are written into the ‘Food.txt’ text file.

Food Collection Function

A screen shot of a computer code

AI-generated content may be incorrect.

*Figure 15 : Food Collection Function*

This Function is used to give out the food ordered online by the customer by checking for the ‘Food ID’ from the data in ‘Food.txt’ text file (FF opens ‘Food.txt’ in a+ mode).

The ‘Food ID’ is attached in the receipt (5-digit code) which finds for the food order using the ID in the text file.

When the ID given by the customer matches with a record in the text file data, it prints that record to the Ticketing Clerk to give out the food and updates the ‘Status’ to “Collected” and ‘Collected At’ with the current timing of the food collection. If no record is found it print “No Food Orders Found”.

The Old data is deleted, and the updated data is written into the text file.

Price Printing Function

A black background with white text

AI-generated content may be incorrect.

*Figure 16 : Price Printing Function*

This Function is used to print out the Movie Ticket and Food Pricing.

Feedback Function

A black screen with yellow text

AI-generated content may be incorrect.

*Figure 17 : Feedback Function*

This Function asks the user to enter Rating (out of 5 stars) and Feedback Comment (Optional) and store it into ‘Feedback.txt’ text file (FBF opens ‘Feedback.txt’ in ‘a+’ mode.

Reading Feedback Function



*Figure 18 : Reading Feedback Function*

This Function reads all the Feedback Stored in the ‘Feedback.txt’ text file (FBF opens ‘Feedback.txt’ in ‘a+’ mode.

Rewards Function

A computer screen with text and numbers

AI-generated content may be incorrect.

*Figure 19 : Rewards Function*

This code updates a rewards file by first **reading all existing records** into a list of strings (data). It then **counts the user's past non-rewarded instances** (where "Rewards" was "No"). A **new entry** for the user is created, setting their reward status based on the input **r**. Crucially, if the current transaction is a "No" reward, it checks if the total count of "No" rewards (past count + 1) exceed five. If it does, a "Penalty": "Yes" flag is added to the new entry. Finally, the **entire file is overwritten** with the complete, updated list of entries.

Reading Rewards Function

A screenshot of a computer program

AI-generated content may be incorrect.

*Figure 20 : Reading Rewards Function*

The rewards\_read function's purpose is to **scan all historical records** in the rewards file (RF) to find an entry matching the provided user **email**. Once a matching entry is found, it **prioritizes checking for a reward** ("Rewards": "Yes") and, if found, prints a discount code message. If no reward is found, it then **checks for a penalty** ("Penalty": "Yes") and prints a warning message about exceeding the limit for false reports. It also sets a **global variable penal** to "Yes" to signal the penalty status to other parts of the program. Otherwise, it sets the **global variable penal** to "No".

Penalty Function

A screen shot of a computer

AI-generated content may be incorrect.

*Figure 21 : Penalty Function*

The **Penalty()** function is a simple utility that **applies a fixed penalty of RM 50**. It takes an existing **TotalP** amount (the user's current total bill or penalty) and **adds RM 50 to it**. It then prints a confirmation message and displays the **new running TotalP** to the user. This function is likely called when a penalty flag, such as the one detected in the previous rewards\_read function, is triggered.

Technical Issue (Reporting) Function

A computer screen with text on it

AI-generated content may be incorrect.

*Figure 22 : Technical Issue (Reporting) Function*

The **Technical\_Issue()** function is designed for **logging into a new technical support case**. It rigorously **validates user input** for the issue description and the Cinema ID, giving the user up to three attempts for each. If validation fails, the function exits. Upon successful validation, it collects additional details like the time of the issue and a unique issue ID (from an external function issueID). Finally, it organizes all this information into a dictionary and **appends the new issue record to a log file** represented by the file handle TF.

Reading Technical Issues

A black screen with green text

AI-generated content may be incorrect.

*Figure 23 : Reading Technical Issues*

The **Technir()** function serves as a **reader for the technical issue log**. It accesses the file handle TF, reads all lines from the file, and prints each entry to the console after stripping trailing whitespace. This allows an administrator or user to view the full list of reported technical issues in raw format. The function does not parse the entries or store them in a variable — it simply displays them for review.

Editing Technical Issues

A screen shot of a computer program

AI-generated content may be incorrect.

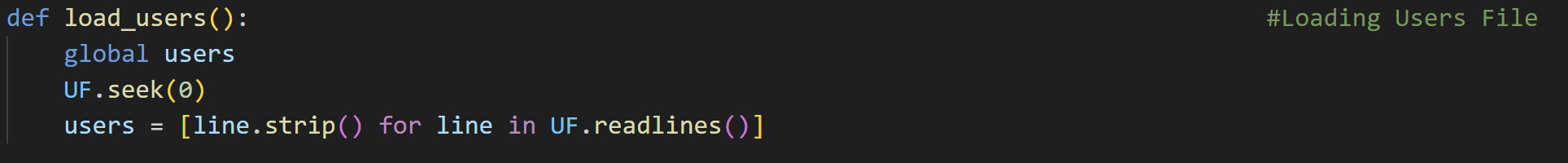
A computer screen shot of code

AI-generated content may be incorrect.

*Figure 24 : Editing Technical Issues*

The Techniw() function is designed to **update technical issue records**. It first asks the user for the **Issue ID** of the record they want to modify. It then prompts the user to choose whether they want to **Update (1)**, **Fix (2)**, or mark it as **No Issue (3)**. If the user chooses to **Update (1)**, they can change a specific field in the record. If they choose to **Fix (2)**, the issue's status is automatically set to "Fixed," and a timestamp is recorded. Similarly, choosing **No Issue (3)** sets the status to "No Issue" and records a timestamp. Finally, after the update, the function overwrites the original data file (TF) with the modified list of issues. Essentially, Techniw() is the part of the program responsible for **managing and modifying the details of existing technical problems.**

**Load Users Function**

****

***Figure 25 : Load Users Function***

The load\_users() function reads all user records from the file handle UF and stores them in the global list users. It begins by resetting the file pointer to the start using UF.seek(0), then reads each line, strips trailing whitespace, and stores the cleaned entries in the users list. This allows the program to access all user records in memory for later use.

**Login Function**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**A screen shot of a computer program

AI-generated content may be incorrect.**

***Figure 26 : Login Function***

The login function verifies the users by checking their email and password. It first loads the list of registered users with load\_users(). After declaring email as global, it asks the user to enter their email and using .strip() trims extra spaces and converts it to lowercase using lower(). Inside an if-loop of emails ending with "gmail.com" or "yahoo.com", the program checks if the email exists in the user records, and the while loop gives the user up to three chances to enter the correct password. The function checks for empty or too-short passwords and gives error messages if the input is invalid. If the password matches, it confirms a successful login. If it doesn’t after three attempts, the login is blocked. If no account is found for the email, the user is offered the option to create a new account. If the email format is invalid, the function immediately rejects it (returns False).

New User Function

**A screen shot of a computer screen

AI-generated content may be incorrect.**

*Figure 27 : New User Function*

The new\_user() function registers a new user by creating an email–password account and storing it. It starts by calling load\_users() to ensure the current list of users is up to date. Then it prompts the user to enter their email, trims any extra spaces using .strip(), and converts it to lowercase using .lower(). The program checks whether the email already exists in the users list. If it does, the function immediately stops and alerts the user. If the email is new, a while loop begins, giving the user up to three attempts to set a valid password. The function requires the password to be at least six characters long and asks the user to re-enter it for confirmation. If both entries match, it creates a dictionary-like string with email and password, then appends this entry to the file using UF.write(). If the passwords don’t match after three tries, the program stops.

**Display Movie Function**

**A computer code on a black background

AI-generated content may be incorrect.**

***Figure 28 : Display Movie Function***

The display movies() function shows all movies available on a specific date. First, it resets the file pointer to the start of the movie file with STF.seek(0) and then reads all the lines in the file using readlines(). The user is then prompted to enter a date in the format DD-MM-YYYY. The line “for line in lines” loops through every saved movie show, examines its date and seat availability, and displays only those shows that match the date is looking. If the dates match and the number of seats available is greater than zero (checking if seats are available), it prints the movie’s details.

**Booking Seats Function**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**A computer screen shot of text

AI-generated content may be incorrect.**

***Figure 29 : Booking Seats Function***

The Booking seat layout function displays the seating arrangement for a selected show and allows users to book seats. It reads the show data from the STF file and searches for the line containing the selected mid (Show ID). It then takes the relevant line to extract important information like the cinema name and seating capacity. Next, it loads existing seat booking data from Seats.txt. If no record exists for the current show, it initializes a new layout with all seats marked as “O” (open). After loading the seat data, the function prints a visual layout of the seats. Then, the user is prompted to enter their seat selections in a format. If the user’s input is valid and matches the required number of tickets (nop is number of people), the selected seats are marked as booked by replacing O with X in the layout. The updated layout is then saved back to Seats.txt so that other users will see the booked seats as unavailable. If the input is invalid, the user has up to three attempts to correct it.

**Seats Available Function**

**A screen shot of a computer program

AI-generated content may be incorrect.**

***Figure 30 : Seats Available Function***

The seats\_available() function checks whether there are enough seats left for a specific show. It reads all the show data from the file and finds the one that matches the given mid (Show ID). If the number of available seats is less than the number of tickets requested (nop is number of people), it informs the user that there aren’t enough seats. If there are enough, it updates the available seat count by subtracting the requested number of tickets, then calls the booking \_seat layout() to let the user select their exact seats. In the else block it append that updated data and calls the booking seat function. Finally, using .seek(0) it goes to the start of the file and with .write, rewrites the updated seat data back to the file to keep the records accurate.

**Seats Available (Cancellation) Function**

A screen shot of a computer

AI-generated content may be incorrect.

*Figure 31 : Seats Available (Cancellation) Function*

The cancel\_seats\_available() function updates **the number of available seats when a booking is cancelled**. It starts by resetting the file pointer to the beginning of the STF file and reading all the show data stored there. Then it creates an empty list ul to store the updated lines. For each line, it checks whether the line contains the matching show ID (x). If it does, it adds that line to rebuild it into a dictionary. These pairs are stripped of spaces and quotes using strip() and then stored in a movie dictionary. It then retrieves the Seats Available value from this dictionary, converting it to an integer. If there are any problems in reading the number, it goes to 0. The function then adds back the number of cancelled tickets (y) to the available seats and ensures it doesn’t drop below zero. It updates the Seats Available with the new value and rebuilds the line into its original string format. The modified line is added to ul, while all other lines remain unchanged. Finally, the function clears the STF file and writes the updated list of lines back into it.

Ticket Numbers Function

A computer screen shot of a program code

AI-generated content may be incorrect.

*Figure 32 : Ticket Numbers Function*

The ticket\_numbers() function starts the ticket booking process by calculating the number of tickets needed for a particular movie. It begins by asking the user to input the show ID(which corresponds to the movie they want to book seats for). The count gives the user up to three attempts to enter valid numbers for the ticket categories: Adults, Children and Senior Citizens or OKU members.

The function uses a try-except block to handle cases where the user enters something other than a number (letters or symbols) It also checks that at least one adult ticket is being purchased. If the user fails three times to provide valid inputs, function is stopped with a message and exits.

Once valid numbers are entered, the total number of tickets (nop is number of people) is calculated. Then the function calls( in sequence), OR\_Bookings(nop), seats\_available(mid) (to find if the seats are available for the number of people booking) and Price(Adult, Child, SaO) to calculate the total ticket price based on the number of tickets in each category.

Save Bookings Function

A computer screen shot of text

AI-generated content may be incorrect.*Figure 33 : Save Bookings Function*

The save bookings function **records the confirmed booking information** into a text file, ensuring that each reservation is saved for future reference. It starts by resetting the file pointer of STF to the beginning and reading all the show data Then, it generates a timestamp with the current date and time, and a unique booking ID using external function. The function then loops through the file’s lines to find the show that matches the current mid (Show ID).Once it finds the correct line, it cleans and splits the line to rebuild it into a dictionary (cn) containing show details like cinema name, hall number. With this, it creates a new entry dictionary containing all booking details. Finally, it formats this dictionary back into a string, keeping the list format for seats, and writes it to the BF file, which stores all booking records

Booking History Function

A computer screen with text on it

AI-generated content may be incorrect.

*Figure 34 : Booking History Function*

The booking\_hist() function displays a user’s past bookings by retrieving and showing all their saved reservation records. BF.seek(0) resets the file pointer to the beginning and reads all the lines. A counter-variable record is used to keep track of how many matching bookings are found. The function loops through each line in the file and checks if the line contains the current user’s email. This check is done using multiple conditions to match different possible formats .If a match is found, the function prints that line, which represents a past booking, and increases the records count. After the loop, if no bookings were found, it informs the user .Otherwise, it will proceed to handle cases where bookings are found.

Book Tickets Function

A computer screen shot of text

AI-generated content may be incorrect.

*Figure 35 : Book Tickets Function*

The book\_tickets() function manages the entire ticket booking process from start to finish. It initializes by calling display\_movies() to show the list of available shows so the user can choose one. Then it calls ticket\_numbers() to collect the number of tickets they want and the show ID. After that, the function asks if the user has any discount codes and applies them by calling Discounts() if they do Next it checks whether the user wants to order food. For yes it sets the order mode to “Online” and calls the Food() function to handle the order. After food selection, the function proceeds to payment by calling Pay(TotalP). If the user chooses “Cash” as the payment method, the function ends there. Otherwise, it records the transaction in the system using OR\_Revenue(TotalP), saves the booking details through save\_bookings(), and finally generates a booking receipt with receipt().

Cancel Booking Function

A screen shot of a computer program

AI-generated content may be incorrect.

*Figure 36 : Cancel Booking Function*

The cancel\_booking() function allows users to cancel an existing movie ticket booking. It starts with moving the file pointer of BF to the start and reading all booking records and iterates through each line in the file, checking whether the line belongs to the user by matching the email. If it does, the line is displayed so the user can see their bookings, and the program asks for the Booking ID they want to cancel. Once the entered booking ID matches, the function first checks whether the booking has already been cancelled by looking for the word Cancelled in the line. If it’s already cancelled, it informs the user and the program stops. Otherwise, it proceeds to cancel the booking. The line is then parsed into key–value pairs by splitting the string and storing them in a dictionary named entry. Using this dictionary, the function retrieves the show ID and number of tickets booked to update the available seats through the cancel\_seats\_available function. It also calls cancel\_OR\_Payment and cancel\_OR\_Revenue to reverse any payment and revenue records related to this booking. This updated line is appended to the ul list. If the line doesn’t match the user’s booking, it is simply added to ul unchanged. Finally, the original file is cleared, and all updated records from ul are written back, ensuring the cancellation is permanently saved. A message is displayed to confirm that the booking has been successfully cancelled.

Receipt Function

A screen shot of a computer program

AI-generated content may be incorrect.

*Figure 37 : Receipt Function*

The receipt() function is used to generate and print a formatted booking receipt for the customer after successful payment. The function begins by accessing the booking data stored in the STF file.

It searches for the booking record that matches the current Show ID (mid). Once a matching line is found, the function splits the text representation of the booking dictionary into key-value pairs and stores them in a local dictionary called cn. The receipt is then formatted into a structured layout displaying:

* Date and time of booking
* Customer email
* Movie name and date
* Cinema hall and location
* Number of tickets and selected seats
* Total amount and payment method
* Booking ID
* Food ID (if applicable)

The receipt is printed for the customer and saved locally as a text file using a filename based on the customer’s email and booking date. This function ensures that every completed booking has an official record for customer reference and audit purposes.

Cash Payment Function

A screen shot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

*Figure 38 : Cash Payment Function*

The CashP() function handles cash payments at the cinema counter. It reads payment data from the CPF file and processes each entry line by line. The ticketing clerk is prompted to input a Payment ID, and the function locates the matching payment record.

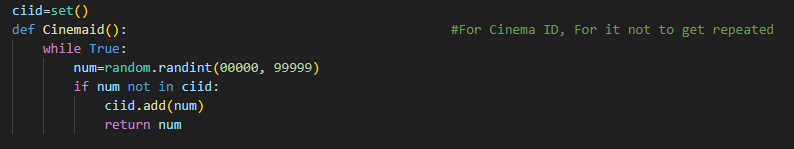
Once the payment record is found, the line is processed and converted into a dictionary format for updating. The clerk is then asked to confirm whether the payment has been made:

* If the customer has paid (Y), the status is updated to "Paid" and the receipt is generated by calling receipt().
* If payment is not made (N), the booking is cancelled, and the status is updated to "Cancelled".

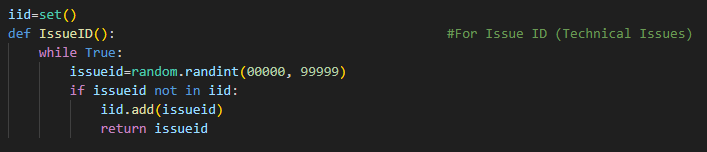
All changes are stored in a new list (updated\_lines) and written back into the CPF file using truncate and rewrite operations. This ensures payment records remain accurate and synchronized with booking status. The function also calls save\_bookings() after a successful payment to update all related booking files.

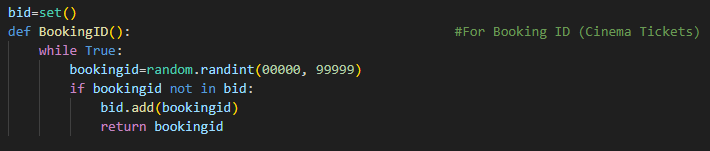
Unique ID Number Functions

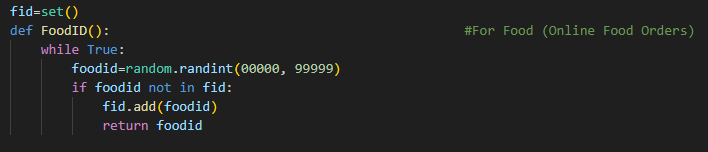
These Functions are used to generate unique numbers for IDs. Sets are used as they don’t store duplicated numbers.

*Figure 39 : Cinema ID Function (Unique ID Number)*

 *Figure 40 : Payment ID Function (Unique ID Number)*

*Figure 41 : Issue ID Function (Unique ID Number)*

*Figure 42 : Booking ID Function (Unique ID Number)*

*Figure 43 : Food ID Function (Unique ID Number)*

Text Files

A screen shot of a computer

AI-generated content may be incorrect.

*Figure 44 : Show Timings.txt*

A screenshot of a computer

AI-generated content may be incorrect.

*Figure 45 : Bookings.txt*

A screenshot of a computer

AI-generated content may be incorrect.

*Figure 46 : Cash Payment.txt*

A screenshot of a computer

AI-generated content may be incorrect.

*Figure 47 : Food.txt*

A screenshot of a computer

AI-generated content may be incorrect.

*Figure 48 : Issues.txt*

A screenshot of a computer

AI-generated content may be incorrect.

*Figure 49 : Feedback.txt*

A screenshot of a computer

AI-generated content may be incorrect.

*Figure 50 : Overall Report.txt*

A screenshot of a computer

AI-generated content may be incorrect.

*Figure 51 : Seats.txt*

A screenshot of a computer

AI-generated content may be incorrect.

*Figure 52 : Payment.txt*

A screenshot of a computer

AI-generated content may be incorrect.

*Figure 53 : Rewards.txt*

A screenshot of a computer

AI-generated content may be incorrect.

*Figure 54 : Users.txt*

Runtime Operation

Register New User

A black screen with white text

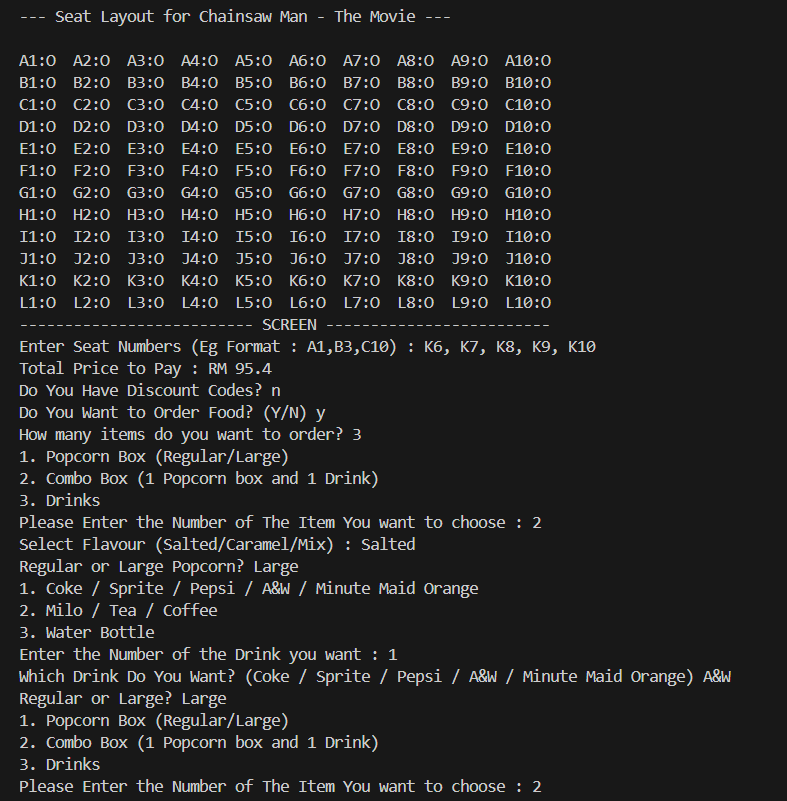
AI-generated content may be incorrect.

*Figure 55 : Register New User*

Login and Booking Sample 1 (Card Payment & Food Order)

A black screen with white text

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.A screen shot of a computer

AI-generated content may be incorrect.

*Figure 56 : Login & Booking Sample 1 (Card Payment & Food Order)*

A screenshot of a computer

AI-generated content may be incorrect.

*Figure 57 : Receipt File for Booking Sample 1 (Card Payment & Food Order)*

Login and Booking Sample 2 (Online Banking Payment)

A black screen with white text

AI-generated content may be incorrect.A screenshot of a computer screen

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

*Figure 58 : Login & Booking Sample 2 (Online Banking Payment)*

A screenshot of a computer

AI-generated content may be incorrect.

*Figure 59 : Receipt File for Booking Sample 2 (Online Banking Payment)*

Login and Booking Sample 3 (with Incorrect Password) (E-Wallet Payment)

A black screen with white text

AI-generated content may be incorrect. A screenshot of a computer

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

*Figure 60 : Login & Booking Sample 3 (with Incorrect Password) (E-Wallet Payment)*

A screenshot of a computer

AI-generated content may be incorrect.

*Figure 61 : Receipt File for Booking Sample 3 (with Incorrect Password) (E-Wallet Payment)*

Login and Booking Sample 4 (Cash Payment)

A black screen with white text

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

*Figure 62 : Login & Booking Sample 4 (Cash Payment)*

View Booking History

A black screen with white text

AI-generated content may be incorrect.

*Figure 63 : View Booking History*

Viewing Pricing

A screen shot of a computer

AI-generated content may be incorrect.

*Figure 64 : View Pricing*

Reporting a Technical Issue

A screen shot of a computer

AI-generated content may be incorrect.

*Figure 65 : Reporting a Technical Issue*

Feedback

A screen shot of a computer

AI-generated content may be incorrect.

*Figure 66 : Giving Feedback*

Technician Reading Issues File

A black screen with white and blue lights

AI-generated content may be incorrect.

*Figure 67 : Technician Reading Issues File*

Technician Updating Issues File (False Issue)

A screen shot of a computer

AI-generated content may be incorrect.

*Figure 68 : Technician Updating Issues File*

Ticketing Clerk (Cash Payment)

A computer screen shot of a black screen

AI-generated content may be incorrect.

*Figure 69 : Ticketing Clerk (Cash Payment)*

Ticketing Clerk (In Store Food Order)

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer screen

AI-generated content may be incorrect.

*Figure 70 : Ticketing Clerk (In Store Food Order)*

Ticketing Clerk (Food Collection)

A black screen with white text

AI-generated content may be incorrect.

*Figure 71 : Ticketing Clerk (Food Collection)*

Cinema Manager (Adding Movie Listing)

A screen shot of a computer

AI-generated content may be incorrect.

*Figure 72 : Cinema Manager (Adding Movie Listing)*

Cinema Manager (Updating Movie Listing)

A screen shot of a computer

AI-generated content may be incorrect.

*Figure 73 : Cinema Manager (Updating Movie Listing)*

Cinema Manager (Removing Movie Listing

A screen shot of a computer

AI-generated content may be incorrect.

*Figure 74 : Cinema Manager (Removing Movie Listing)*

Cinema Manager (View Overall Report)

A screen shot of a computer

AI-generated content may be incorrect.

*Figure 75 : Cinema Manager (View Overall Report)*

Cinema Manager (View Feedback)

A screen shot of a computer

AI-generated content may be incorrect.

*Figure 76 : Cinema Manager (View Feedback)*

Assumptions Of System Developed

**User Roles Are Predefined**

The system supports distinct roles: Customer, Technician, Ticket Clerk, and Cinema Manager.

Role access is determined by login credentials or hidden ID prompts.

**File-Based Data Storage**

All data (users, bookings, payments, issues, rewards, etc.) is stored in plain text files.

No external database or cloud storage is used.

**Email Is the Unique Identifier**

Each user is identified by their email address.

Email is used to link bookings, rewards, and feedback.

**Food Orders Are Optional and Linked to Bookings**

Food can be ordered online or in-store.

Online food orders are limited to the number of seats booked.

**Payment Is Mandatory Before Booking Is Confirmed**

Payment methods include Card, Online Banking, E-Wallet, and Cash.

Cash payments require in-store confirmation via a unique Payment ID.

**Technical Issues Can Be Reported Only After Login**

Users can report screening-time issues with a retry limit.

False reports may trigger penalties after repeated misuse.

**Discounts and Penalties Applied Dynamically**

Valid discount codes reduce ticket prices.

Penalties are added for users flagged by the technician module.

**Booking Is Seat-Specific**

Users must select seats manually.

Duplicate or invalid seat entries are rejected.

**All Operations Are Menu-Driven**

The system runs in a console environment with numbered options.

Each function is triggered based on user input.

**No Real-Time Concurrency**

The system assumes single-user access at a time.

No locking or concurrency control is implemented.

**Audit and Reporting Are Cumulative**

Revenue and seat bookings are tracked cumulatively in Overall Report.txt.

Cancellations trigger rollback of revenue and seat counts.

Advantages Of System Developed

**Multi-Role Access**

Supports distinct roles: Customer, Technician, Ticket Clerk, and Cinema Manager.

Each role has tailored functionality, improving operational clarity and control.

**Input Validation**

All user inputs are validated with retry limits.

Prevents crashes and ensures clean data entry across modules.

**Dynamic Seat Management**

Real-time seat layout visualization and booking.

Prevents double-booking and enforces seat format validation.

**Multiple Payment Integration**

Supports Card, Online Banking, E-Wallet, and Cash.

Cash payments are deferred and linked to in-store confirmation.

**Food Ordering System**

Integrated food menu with pricing and GST.

Supports both online and in-store orders, linked to seat count.

**Technical Issue Reporting**

Users can report screening-time issues with retry limits.

Technician module allows updates, resolution tracking, and reward assignments.

**Comprehensive Reporting**

Tracks total revenue, seats booked, and payment method usage.

Supports rollback on cancellations for accurate metrics.

**Modular and Maintainable Codebase**

Each function is isolated, and purpose driven.

Easy to extend, debug, or refactor for future enhancements.