System Requirements Specification Index

For

Music Festival Console

Version 1.0

IIHT Pvt. Ltd.

fullstack@iiht.com

TABLE OF CONTENTS

Pizza Shop Calculator Console

System Requirements Specification

1 PROJECT ABSTRACT

Due to a surge in international artists touring India, BookMyShow, India's leading entertainment ticketing platform, requires a robust solution to efficiently manage their concert venue operations. The Concert Management System is a Python console application designed to streamline the complex task of managing large-scale concert events. This system helps venue managers track real-time ticket sales, monitor seating availability, and calculate revenue across premium, middle, and rear seating zones. The system performs critical functions including dynamic pricing calculations for different seating zones, real-time tracking of seat occupancy and availability, efficient management of seating arrangements, and generation of comprehensive sales and occupancy statistics. By automating these calculations and providing instant analytics, the system ensures accurate tracking for both sales monitoring and seating management, helping BookMyShow maintain its high standards of service for both artists and concert-goers.

BUSINESS REQUIREMENTS:

Screen Name	Console input screen
Problem Statement	1. User needs to track ticket sales and calculate revenueThe application should handle discounts and split billing 2. Application should monitor available seats and occupancy 3. Console should handle different calculations using various arithmetic operators:- Multiplication (*) for calculating zone revenue, Addition (+) for combining total revenue, Subtraction (-) for calculating remaining seats, Division (/) for calculating occupancy percentages, Floor Division (//) for calculating complete rows, Modulus (%) for calculating remaining seats in rows

2 CONSTRAINTS

2.1 INPUT REQUIREMENTS

1. Zone A Tickets:
    - **Must be stored as integer in variable zone_a_sold**
    - **Must be non-negative and not exceed 200**
    - **Example: 150**
2. Zone B Tickets:
    - Must be stored as integer in variable zone_b_sold
    - Must be non-negative and not exceed 300
    - Example: 200
3. Zone C Tickets:
    - Must be stored as integer in variable zone_c_sold
    - Must be non-negative and not exceed 500
    - Example: 350
4. Zone Capacities:
    - Must be stored as integer in quantity
    - Must be stored as constants ZONE_A_CAPACITY = 200
    - Must be stored as constants ZONE_B_CAPACITY = 300
    - Must be stored as constants ZONE_C_CAPACITY = 500
    - Must be stored as constants SEATS_PER_ROW = 20
5. Zone Prices:
    - Must be stored as constants ZONE_A_PRICE = 5000
    - Must be stored as constants ZONE_B_PRICE = 3000
    - Must be stored as constants ZONE_C_PRICE = 1500

2.2 CALCULATION CONSTRAINTS

1. Revenue Calculation:
    - Use multiplication (*) for zone revenue
    - Use addition (+) for total revenue
    - Store in total_revenue
    - Example: (150 * 5000) + (200 * 3000) + (350 * 1500)
2. Remaining Seats:
    - Use multiplication (*) *and division (/) for percentage*
    - Use subtraction (-)
    - Store results in zone_a_left, zone_b_left, zone_c_left
    - Example: 200 - 150 = 50 seats left
3. Occupancy Percentage:
    - Use multiplication (*) and division (/)
    - Store results in a_occupancy, b_occupancy, c_occupancy
    - Example: (150 * 100) / 200 = 75.0%
4. Row Distribution:
    - Use floor division (//) for complete_rows
    - Use modulus (%) for remaining_seats
    - Example: 150 // 20 = 7 rows, 150 % 20 = 10 seats

2.3 Output Constraints

1. Display Format:
    - Show revenue with ₹ symbol and comma separators
    - Show occupancy percentages with one decimal place
    - Each value must be displayed on a new line

- Use simple section headers with hyphen separators - Group information by zone

2. Required Output Format:
    - Must show exactly in this order:
    - Show "Sales Summary"
    - Show "Total Revenue: ₹{value}"
    - Show "Seating Status"
    - Show "Zone A:"
        - Show "Remaining Seats: {value}"
        - Show "Occupancy: {value}%"
        - Show "Complete Rows: {value}"
        - Show "Extra Seats: {value}"
    - Show "Zone B:"
        - Show "Remaining Seats: {value}"
        - Show "Occupancy: {value}%"
        - Show "Complete Rows: {value}"
        - Show "Extra Seats: {value}"
    - Show "Zone C:"
        - Show "Remaining Seats: {value}"
        - Show "Occupancy: {value}%"
        - Show "Complete Rows: {value}"
        - Show "Extra Seats: {value}"

4. TEMPLATE CODE STRUCTURE:

1. Calculation Functions:
    - calculate_ticket_revenue() [*, +]
    - calculate_seats_remaining() [-]
    - calculate_zone_occupancy() [*, /]
    - calculate_seats_per_row() [//, %]
2. Input Section:
    - Get Zone A tickets sold (int)
    - Get Zone B tickets sold (int)
    - Get Zone C tickets sold (int)
3. Conversion Section:
    - Calculate total revenue
    - Calculate remaining seats
    - Calculate occupancy percentages
    - Calculate row distributions
4. Output Section:
    - Display total revenue
    - Display seating status for each zone
    - Format all numbers appropriately

5. DETAILED FUNCTION IMPLEMENTATION GUIDE

5.1 Revenue Calculation Functions

1. Write a Python function to calculate total ticket revenue from all zones. Define: `calculate_ticket_revenue(zone_a_sold, zone_b_sold, zone_c_sold)`

The function should:

- Accept three integer parameters representing tickets sold in each zone
- Define constants for zone pricing: ZONE_A_PRICE = 5000, ZONE_B_PRICE = 3000, ZONE_C_PRICE = 1500
- Validate that all inputs are integers using `isinstance()` and `all()`
- Validate that all ticket counts are non-negative (>= 0)
- Raise `ValueError` with message "Number of tickets must be whole numbers" for non-integer inputs
- Raise `ValueError` with message "Number of tickets cannot be negative" for negative inputs
- Calculate zone revenues using multiplication (*): `zone_a_revenue = zone_a_sold * ZONE_A_PRICE`
- Calculate total revenue using addition (+): `total_revenue = zone_a_revenue + zone_b_revenue + zone_c_revenue`
- Return the total revenue as an integer
- Example: calculate_ticket_revenue(150, 200, 350) should return 2,175,000

5.2 Seating Management Functions

2. Write a Python function to calculate remaining seats in each zone. Define: `calculate_seats_remaining(zone_a_sold, zone_b_sold, zone_c_sold)`

The function should:

- Accept three integer parameters representing tickets sold in each zone
- Define constants for zone capacities: ZONE_A_CAPACITY = 200, ZONE_B_CAPACITY = 300, ZONE_C_CAPACITY = 500
- Validate that all inputs are integers and non-negative
- Raise `ValueError` for invalid input types or negative values
- Check that sold tickets do not exceed zone capacities
- Raise `ValueError` with message "Tickets sold cannot exceed zone capacity" if any zone is oversold
- Calculate remaining seats using subtraction (-): `zone_a_left = ZONE_A_CAPACITY - zone_a_sold`
- Return a tuple containing remaining seats for all zones: `(zone_a_left, zone_b_left, zone_c_left)`
- Example: calculate_seats_remaining(150, 200, 350) should return (50, 100, 150)

3. Write a Python function to calculate zone occupancy percentage. Define: `calculate_zone_occupancy(zone_sold, zone_capacity)`

The function should:

- Accept two integer parameters: tickets sold and zone capacity
- Validate that both inputs are integers using `isinstance()`
- Raise `ValueError` with message "Values must be whole numbers" for non-integer inputs
- Validate that zone_sold is non-negative (>= 0)
- Validate that zone_capacity is positive (> 0)
- Raise `ValueError` with message "Tickets sold cannot be negative" for negative sold tickets
- Raise `ValueError` with message "Capacity must be positive" for zero/negative capacity
- Check that sold tickets do not exceed capacity
- Raise `ValueError` with message "Sold tickets cannot exceed capacity" if oversold
- Calculate occupancy using multiplication (*) and division (/): `(zone_sold * 100) / zone_capacity`
- Return the occupancy percentage as a float
- Example: calculate_zone_occupancy(150, 200) should return 75.0

5.3 Row Distribution Functions

4. Write a Python function to calculate complete rows and remaining seats. Define: `calculate_seats_per_row(total_seats)`

The function should:

- Accept one integer parameter representing total seats
- Define constant: SEATS_PER_ROW = 20
- Validate that input is an integer using `isinstance()`
- Raise `ValueError` with message "Seats must be a whole number" for non-integer input
- Validate that total_seats is non-negative (>= 0)
- Raise `ValueError` with message "Seats cannot be negative" for negative input
- Calculate complete rows using floor division (//): `complete_rows = total_seats // SEATS_PER_ROW`
- Calculate extra seats using modulus (%): `remaining_seats = total_seats % SEATS_PER_ROW`

- Return a tuple containing both values: `(complete_rows, remaining_seats)`
- Example: calculate_seats_per_row(45) should return (2, 5) meaning 2 complete rows and 5 extra seats

5.4 Main Program Function

5. Write a Python main program to demonstrate the concert management system.
Define: `main()` function or use `if __name__ == "__main__":` block

The function should:

- Print system header: "Concert Management System"
- Get user input for tickets sold in each zone using `int(input())`
    - Prompt: "Enter Zone A tickets sold: "
    - Prompt: "Enter Zone B tickets sold: "
    - Prompt: "Enter Zone C tickets sold: "
- Calculate total revenue using `calculate_ticket_revenue()`
- Calculate remaining seats using `calculate_seats_remaining()` and unpack the tuple
- Calculate occupancy for each zone separately using `calculate_zone_occupancy()` with proper capacities:
    - Zone A: calculate_zone_occupancy(zone_a_sold, 200)
    - Zone B: calculate_zone_occupancy(zone_b_sold, 300)
    - Zone C: calculate_zone_occupancy(zone_c_sold, 500)
- Calculate row distribution for each zone using `calculate_seats_per_row()` and unpack tuples
  Display results in this exact format:
  Sales Summary

  Total Revenue: ₹{total_revenue}

  Seating Status

  Zone A:

  Remaining Seats: {a_left}

  Occupancy: {a_occupancy}%

  Complete Rows: {a_rows}

  Extra Seats: {a_extra}

  Zone B:

  Remaining Seats: {b_left}

Occupancy: {b_occupancy}%Complete

Rows: {b_rows}

Extra Seats: {b_extra}

Zone C:

Remaining Seats: {c_left}

Occupancy: {c_occupancy}%Complete

Rows: {c_rows}

Extra Seats: {c_extra}

- Use proper variable names: total_revenue, a_left, b_left, c_left, a_occupancy, b_occupancy, c_occupancy
- Format revenue with ₹ symbol and display occupancy percentages as returned by the function
- Handle all user inputs and function calls properly
- Example execution should work with sample inputs like: Zone A=150, Zone B=200, Zone C=350

6. EXECUTION STEPS TO FOLLOW:

1. Run the program

2. Enter Zone A tickets sold

3. Enter Zone B tickets sold

4. Enter Zone C tickets sold

5. View Sales Summary showing total revenue

6. View Seating Status showing:
   - Remaining seats per zone
   - Occupancy percentage per zone
   - Row distribution per zone

Execution Steps to Follow:

- All actions like build, compile, running application, running test cases will be through Command Terminal.

● To open the command terminal the test takers, need to go to Application menu (Three horizontal lines at left top) -> Terminal -> New Terminal

● This editor Auto Saves the code

● If you want to exit(logout) and continue the coding later anytime (using Save & Exit option on Assessment Landing Page) .

● These are time bound assessments the timer would stop if you logout and while logging in back using the same credentials the timer would resume from the same time it was stopped from the previous logout.

● To launch application: python3 filename.py

● To run Test cases: python3 -m unittest

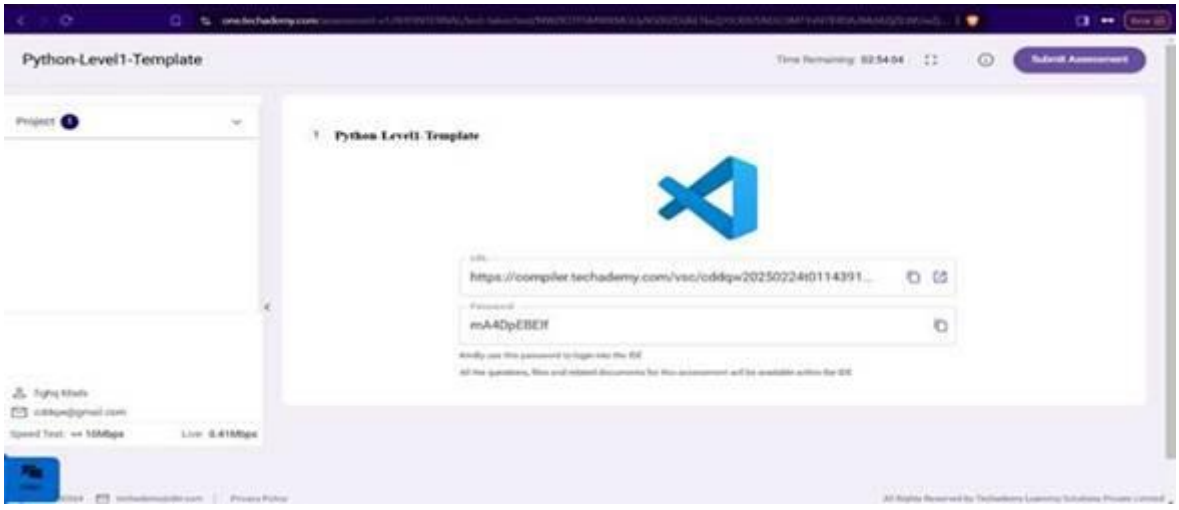
Screen shot to run the program

To run the application

Python3 filename.py

To run the testcase

python3 -m unittest



- Once you are done with development and ready with submission, you may navigate to the previous tab and submit the workspace. It is mandatory to click on “Submit Assessment” after you are done with code.