System Requirements Specification Index

For

Customer Support Ticket Tracking Console Application

Version 1.0

IIHT Pvt. Ltd.
fullstack@iiht.com

TABLE OF CONTENTS

Customer Support Ticket Tracking Console

System Requirements Specification

1 PROJECT ABSTRACT

TechHelp Inc. requires a ticket tracking system for their customer support operations. The system will track customer issues, assign priorities, and monitor resolution status using Python's list operations and methods. This console application demonstrates fundamental list operations while allowing support agents to manage customer requests effectively. The system performs critical functions including categorizing tickets by type, sorting by priority, and tracking resolution progress. By implementing these operations with Python lists, the system provides an efficient way for support teams to organize their workload.

2 BUSINESS REQUIREMENTS:

Screen Name	Console input screen
Problem Statement	1. System needs to store and categorize different types of tickets (billing, technical, feature, account) 2. Application must allow tickets to be sorted by different attributes (priority, id) 3. System must support filtering tickets by type, status, or keyword 4. Console should handle different list operation like Addition (+) for combining ticket queues, Slicing for selecting subset of ticket ,List comprehension for filtering tickets , Advanced list methods (sort, append, pop, clear)

3 CONSTRAINTS

3.1 INPUT REQUIREMENTS

1. Ticket Records:
    - Must be stored as dictionaries with fields for id, title, type, priority, status
    - Must be stored in list variable `tickets`
    - Example: `{"id": "T001", "title": "Payment not processing", "type": "billing", "priority": 2, "status": "open"}`
2. Ticket Types:
    - Must be one of: "billing", "technical", "feature", "account"
    - Must be stored as string in ticket's "type" field
    - Example: "billing"
3. Priority Levels:
    - Must be stored as integer in "priority" field
    - Must be between 1-4 (1=critical, 4=low)
    - Example: 2
4. Ticket Status:
    - Must be one of: "new", "open", "resolved", "closed"
    - Must be stored as string in ticket's "status" field
    - Example: "open"
5. Active Queues:
    - Must be stored as list in variable `active_queue`
    - Must contain references to tickets from main list
    - Must not exceed 5 tickets
    - Example: `[tickets[0], tickets[3]]`
6. Predefined Tickets:
    - Must use these exact predefined tickets in the initial ticket list:
        - `{"id": "T001", "title": "Payment not processing", "type": "billing", "priority": 2, "status": "open"}`
        - `{"id": "T002", "title": "Reset password", "type": "account", "priority": 3, "status": "new"}`

- `{"id": "T003", "title": "Application crashes", "type": "technical", "priority": 1, "status": "open"}`
- `{"id": "T004", "title": "Add dark mode", "type": "feature", "priority": 4, "status": "new"}`
- `{"id": "T005", "title": "Renewal failed", "type": "billing", "priority": 2, "status": "open"}`

7. Escalated Tickets:
   - Must use these exact predefined items in the escalated list:
     - `{"id": "E001", "title": "Security breach", "type": "technical", "priority": 1, "status": "new"}`
     - `{"id": "E002", "title": "Double-charged", "type": "billing", "priority": 1, "status": "new"}`

3.2 Operations Constraints

1. Ticket Queue Combination:
   - Must use exact list addition operator `+`
   - Example: `tickets + escalated` combines current tickets with escalated tickets
2. Ticket Subset Selection:
   - Must use exact `tickets[start:end:step]` notation
   - Example: `tickets[0:5]` selects first 5 tickets
3. Ticket Filtering:
   - Must use list comprehension
   - Example: `[ticket for ticket in tickets if ticket["type"] == "technical"]`
4. Priority Queue Management:
   - Must use list indexing and operations
   - Must validate against queue constraints
   - Example: Adding a ticket: `active_queue.append(tickets[3])`
5. Ticket Status Updates:
   - Must use list and dictionary operations to update status
   - Example: `tickets[index]["status"] = "resolved"`

3.3 OUTPUT CONSTRAINTS

1. Display Format:

- - Show ticket ID, title, type, priority, status
- - Format priority with level indicator
- - Format status with appropriate visual indicator
- - Each ticket must be displayed on a new line

2. Required Output Format:

- - Show "===== TICKET TRACKING SYSTEM ====="
- - Show "Total Tickets: {count}"
- - Show "Critical Tickets: {count}"
- - Show "Current Ticket List:"
- - Show tickets with format: "{id} | {title} | {type} | Priority: {priority} | {status}"
- - Show "Active Queue:" when displaying active queue
- - Show "Filtered Results:" when displaying search results

4. TEMPLATE CODE STRUCTURE:

1. Data Management Functions:

- `initialize_data()` - creates the initial ticket and escalated lists
- `add_ticket(tickets, ticket)` - adds a ticket to the list
- `remove_ticket(tickets, index)` - removes a ticket at the specified index

2. List Operation Functions:

- - `sort_tickets(tickets, key)` - sorts tickets by key
- - `filter_tickets(tickets, filter_type, value)` - filters tickets by criteria
- - `combine_queues(tickets1, tickets2)` - combines two ticket lists

3. Queue Management:

- - `manage_queue(tickets, active_queue, operation, index)` - handles queue operations

4. Ticket Management:

- - `update_ticket(tickets, index, field, value)` - updates ticket fields

5. Display Functions:

- - `get_formatted_ticket(ticket)` - formats a ticket for display
- - `display_data(data, data_type)` - displays tickets or queue

6. Program Control:

- - `main()` - main program function

5. EXECUTION STEPS TO FOLLOW:

1. Run the program
2. View the main menu
3. Select operations:
   - Option 1: View Tickets
   - Option 2: Add/Remove Ticket
   - Option 3: Sort Tickets
   - Option 4: Filter Tickets
   - Option 5: Queue Operations
   - Option 6: Process Escalated
   - Option 7: Update Ticket
   - Option 0: Exit
4. Perform operations on the ticket list
5. View results after each operation
6. Exit program when finished