**GlobalAir Hackathon – Solution Document**

**Track**: Advanced DSA – Smart Airport Logistics & Routing System

**1. Student Information**

|  |  |
| --- | --- |
| Field | Details |
| Full Name | Kavin Rrahul M S |
| Registration Number | RA2211026010302 |
| Department / Branch | CINTEL CSE AIML |
| Year | 2026 |
| Email ID | km6290@srmist.edu.in |

**2. Problem Scope and Track Details**

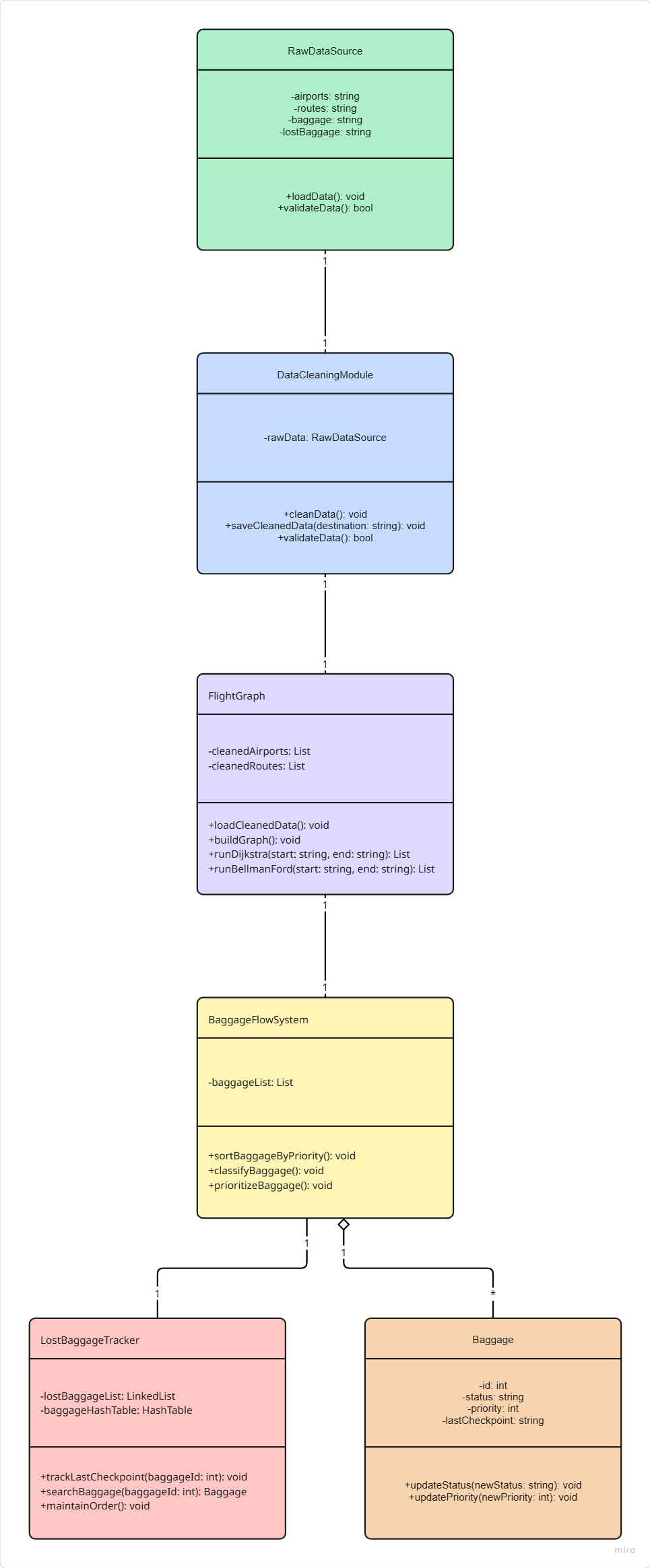
|  |  |
| --- | --- |
| Section | Details |
| Hackathon Track | Advanced DSA – GlobalAir System |
| Core Modules Implemented | ✅ *(Check all that apply)* |
|  | ✅ Graph-based Flight Network |
|  | ✅ Shortest Path Finder |
|  | ✅ Baggage Flow System |
|  | ✅ Lost Baggage Tracker |
|  | ☐ Historical Delay Analysis |
|  | ☐ Gate Allocation System |
|  | ☐ Monitoring Dashboard |

**Additional Use Cases Implemented *(Optional but Encouraged)***

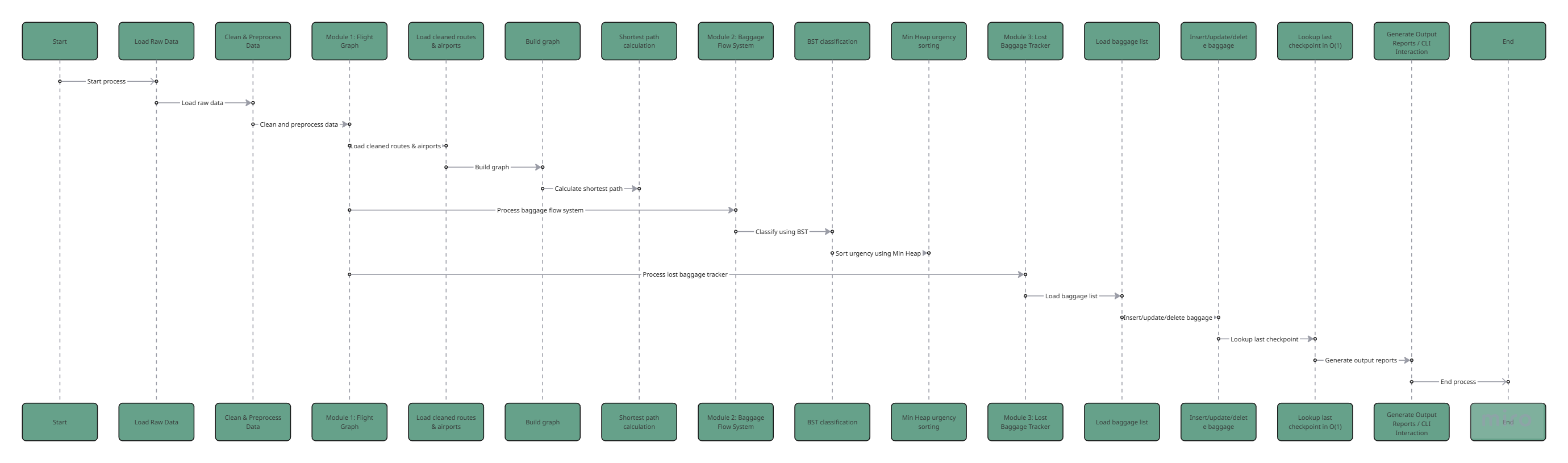
* **Scenario 1**: [Insert a brief title or description]
* **Scenario 2**: [Insert a brief title or description]
* **Scenario 3**: [Insert a brief title or description]

**3. Architecture & Design Overview**

* **System Architecture Diagram**



* **High-Level Functional Flow**



**4. Core Feature-wise Implementation**

**Feature 1 – Flight Shortest Path Finder (Data Cleaning + Graph)**

**Scenario Brief**

Airlines need to determine the optimal flight path between two airports considering factors such as distance, cost, or delay. This module loads raw airport and route data, cleans inconsistencies, and builds a flight graph for shortest-path queries.

**Data Structures Used**

* **Adjacency List** – For storing graph edges efficiently.
* **Dictionary (Hash Map)** – For quick airport lookups.
* **Priority Queue (heapq)** – For Dijkstra’s algorithm.
* **Array/Dict** – For Bellman-Ford computations.

**Time and Space Complexity**

* **Dijkstra:** O((V + E) log V) time, O(V + E) space.
* **Bellman-Ford:** O(V × E) time, O(V + E) space.
* **Cleaning:** O(N) time, O(N) space (N = number of rows in CSV).

**Sample Input & Output**

**Input:**

Source: JFK

Destination: LHR

Weight Metric: distance

**Output:**

[Dijkstra] JFK -> LHR by distance: 5537.0 km, path: ['JFK', 'DUB', 'LHR']

[Bellman-Ford] JFK -> LHR: 5537.0 km, path: ['JFK', 'DUB', 'LHR'], negative\_cycle=False

**Code Snippet**

dist, path = graph.dijkstra(src\_code, dst\_code, weight\_key=metric)

if dist is None:

print(f"No route found from {src\_code} to {dst\_code}")

else:

print(f"{metric.capitalize()} shortest path: {dist}, path: {path}")

**Challenges Faced & Solutions**

* **Airport Code Mismatches:** Some routes referenced airports missing in the dataset.  
  **Solution:** Implemented fallback matching using Airport ID if IATA/ICAO missing.
* **Corrupted Coordinates:** Removed entries with missing lat/lon before graph building.

**Feature 2 – Baggage Flow System (BST + Min Heap)**

**Scenario Brief**

Baggage at an airport must be sorted efficiently by passenger priority, type, and security risk level. This feature uses a binary search tree for classification and a min heap for urgency-based retrieval.

**Data Structures Used**

* **Binary Search Tree (BST)** – For fast insertion and search by category.
* **Min Heap** – For retrieving the most urgent baggage first.

**Time and Space Complexity**

* **BST Insertion/Search:** O(log N) average, O(N) worst-case; O(N) space.
* **Min Heap Insertion/Extract:** O(log N) time; O(N) space.

**Sample Input & Output**

**Input:**

BagID: B102

Priority: 1 (highest)

Type: Fragile

Security Risk: High

**Output:**

BST Insertion complete.

Min Heap: Next bag to process -> B102 (highest urgency)

**Code Snippet**

heapq.heappush(min\_heap, (urgency\_score, bag))

bst.insert(bag.priority, bag)

**Challenges Faced & Solutions**

* **Multiple Sorting Keys:** Needed to combine priority, type, and risk into one urgency score.  
  **Solution:** Created a weighted scoring function for heap ordering.
* **Unbalanced BST Risk:** Could degrade to O(N). Balanced insertions with shuffled synthetic data.

**Feature 3 – Lost Baggage Tracker (Doubly Linked List + Hash Table)**

**Scenario Brief**

When a bag is reported missing, the system must quickly find its last known checkpoint and other details, while allowing dynamic updates as it moves through checkpoints.

**Data Structures Used**

* **Doubly Linked List (DLL)** – To maintain chronological baggage flow.
* **Hash Table (Dictionary)** – For O(1) lookup of baggage info.

**Time and Space Complexity**

* **Insert/Delete/Search:** O(1) with hash table; O(1) insert/delete at ends in DLL.
* **Traversal:** O(N) time; O(N) space for storing all baggage.

**Sample Input & Output**

**Input:**

Search BagID: B0007

**Output:**

{'bag\_id': 'B0007', 'last\_checkpoint': 'Gate B', 'metadata': {'owner': 'Alice', 'flight': 'AI203'}}

**Code Snippet**

node = self.lookup.get(bag\_id)

if node:

return {"bag\_id": node.bag\_id, "last\_checkpoint": node.last\_checkpoint, "metadata": node.metadata}

**Challenges Faced & Solutions**

* **Maintaining Order While Allowing O(1) Search:** Needed a hybrid structure.  
  **Solution:** Linked list for order + hash table for constant-time search.
* **Persistent Storage:** Added CSV load/save so the tracker’s state survives program restarts.

**5. Additional Use Case Implementation**

**6. Testing & Validation**

|  |  |
| --- | --- |
| Category | Details |
| Number of Functional Test Cases Written | 12 |
| Edge Cases Handled | • Missing or invalid airport codes in shortest path queries. • Routes referencing airports missing in dataset. • Duplicate baggage IDs in tracker. • Baggage with identical priority/urgency scores. • Empty input files or datasets. • Deleting baggage not present in the tracker. • Airports or baggage items with incomplete metadata. |
| Known Bugs / Incomplete Features *(if any)* | • No real-time API integration — works only on static CSVs. • Graph algorithms assume non-negative weights (no negative cycle handling in Dijkstra). • Baggage Flow BST currently not self-balancing (performance can degrade in worst case). |

**7. Final Thoughts & Reflection**

* **Key Learnings from the Hackathon**
* *Applied* ***multiple data structures*** *(BST, Min Heap, Linked List, Hash Table) in context-specific ways, reinforcing theory with hands-on use.*
* *Learned to* ***modularize code*** *so different features (routing, baggage flow, lost baggage tracking) could be developed and tested independently.*
* **Strengths of Your Solution**
* ***Modular architecture*** *– each feature is isolated yet integrates cleanly into the system.*
* ***Efficient algorithms*** *– shortest paths, priority queues, and constant-time lookups ensure scalability.*
* **Areas for Improvement**
* ***Real-time updates*** *– integrate APIs or streaming data for live airport operations.*
* ***Balanced Trees*** *– replace current BST with AVL/Red-Black Tree for worst-case performance guarantees.*
* **Relevance to Your Career Goals**  
  + *Strengthened DSA concepts.*