GlobalAir Hackathon – Solution Document Template

Track: Advanced DSA – Smart Airport Logistics & Routing System

---

1. Student Information

Field	Details
Full Name	Meghraj Mahesh Patil
Registration Number	RA221003011683
Department / Branch	CTECH / CSE Core
Year	4th year
Email ID	mp3459@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

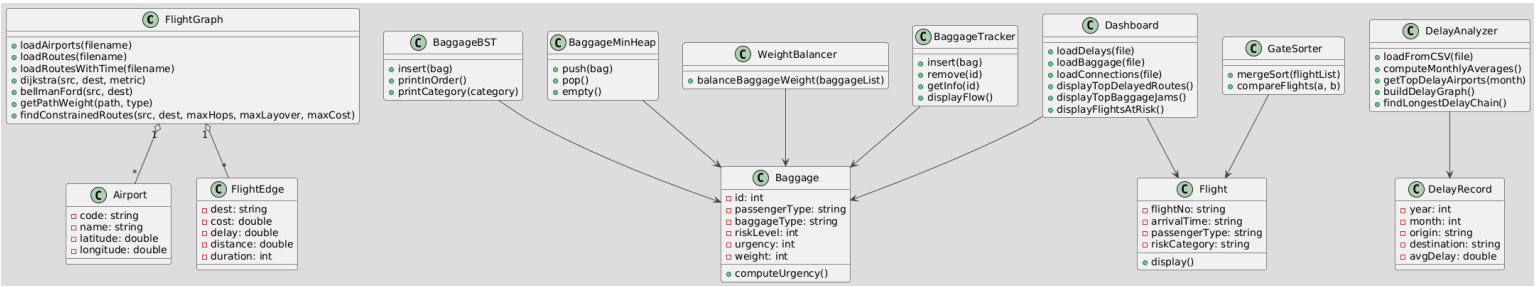
Additional Use Cases Implemented *(Optional but Encouraged)*

- **Scenario 1**: Baggage Weight Balancer (Subset Sum DP)
- **Scenario 2**: Business Route Optimizer (Constrained DFS)

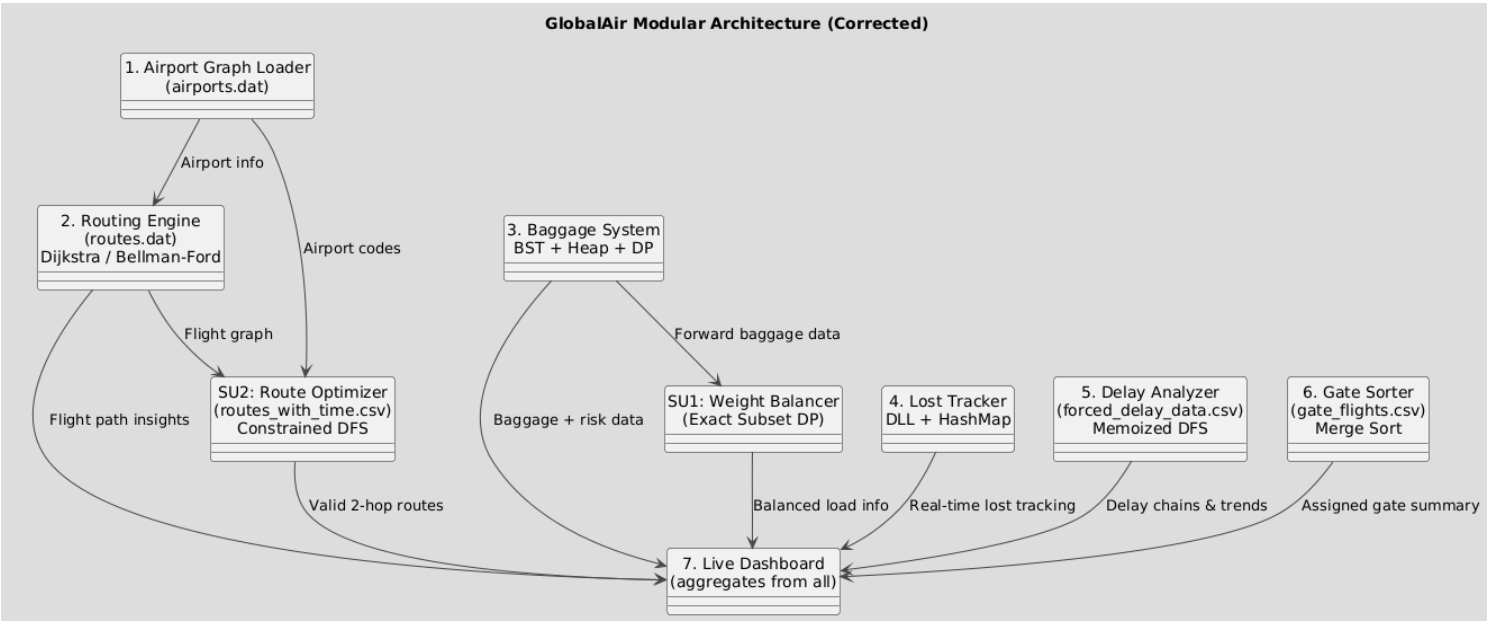
---

3. Architecture & Design Overview

- **System Architecture Diagram**



- **High-Level Functional Flow**



---

4. Core Feature-wise Implementation

Feature: Airport Graph (Module 1)

- **Scenario Brief**
  Parses global airport metadata (`airports.dat`) and maps it to nodes in a graph using standard IATA codes like 'BOM', 'MAA', etc.
- **Data Structures Used**
  `unordered_map<string, Airport>` for airport lookup
  `unordered_map<string, vector<FlightEdge>>` for adjacency list representing flight routes
- **Time and Space Complexity**
  `loadAirports()` → Time: O(n), Space: O(n)
  `displayGraph()` → Time: O(V + E)
- **Sample Input & Output**
  Input: `airports.dat`
  Output:

```
Airports loaded: 6072
Skipped malformed: 1626
```

- **Code Snippet**

```
airports[code] = Airport(code, lat, lon, ...);
adjList[src].push_back({dest, distance, cost, delay});
```

- **Challenges Faced & How You Solved Them**
  Handled malformed lines using validation checks and `.isdigit()` to skip invalid rows.

Feature: Pathfinding – Dijkstra and Bellman-Ford (Module 2)

- **Scenario Brief**
  Calculates the shortest path between two airports based on `cost`, `delay`, or `distance` using Dijkstra and Bellman-Ford algorithms.
- **Data Structures Used**
  `priority_queue` for Dijkstra
  `unordered_map` for storing distances, parents
  Adjacency list from Module 1
- **Time and Space Complexity**
  Dijkstra → Time: O((V + E) log V), Space: O(V + E)
  Bellman-Ford → Time: O(V × E), Space: O(V + E)
- **Sample Input & Output**
  Input: `routes.dat`
  Query: `src = BOM, dest = MAA`
  Output:

```
Dijkstra (delay): BOM → HYD → MAA
Total delay: 93 mins
```

- **Code Snippet**

```
priority_queue<pair<double, string>, vector<>, greater<>> pq;
while (!pq.empty()) {
  auto [currDist, currNode] = pq.top(); pq.pop();
  for (auto& edge : adj[currNode]) {
    if (relaxation possible) pq.push(...);
  }
}
```

- **Challenges Faced & How You Solved Them**
  Delay-based negative weights prevented Dijkstra; Bellman-Ford used as a fallback.

Feature: Baggage System – BST, MinHeap, Weight Balancer (Module 3)

- **Scenario Brief**
  Manages baggage priority, flow, and balancing across compartments.
- **Data Structures Used**
  BST for sorted baggage access

MinHeap for urgency-based processing
2D DP Table for optimal weight balancing

- **Time and Space Complexity**
  BST Insert → O(log n) avg, O(n) worst
  MinHeap Push/Pop → O(log n)
  Weight Balance DP → O(n × W)
- **Sample Input & Output**
  Input: 5 test bags with weights
  Output:

```
Compartment A (Weight: 62): IDs = 101, 103
Compartment B (Weight: 60): IDs = 102, 104, 105
```

- **Code Snippet**

```
dp[i][w] = dp[i-1][w] || dp[i-1][w - weight[i]];
```

- **Challenges Faced & How You Solved Them**
  Implemented exact DP instead of greedy to prevent small imbalances in total weight.

Feature: Lost Baggage Tracker (Module 4)

- **Scenario Brief**
  Real-time tracking of baggage items with O(1) lookup and deletion.
- **Data Structures Used**
  Doubly Linked List for order
  HashMap for fast lookup by ID
- **Time and Space Complexity**
  Insert/Delete/Search → O(1)
  Print Flow → O(n)
- **Sample Input & Output**
  Input: Insert Bag ID 103, then remove it
  Output:

```
Bag ID 103 removed
Updated flow: 101 → 102 → 104 → 105
```

- **Code Snippet**

```
unordered_map<int, Node*> bagMap;
bagMap[id] = new Node(baggage);
```

- **Challenges Faced & How You Solved Them**
  Maintained bidirectional pointer consistency while deleting mid-list elements.

Feature: Delay Analyzer – Seasonal + Longest Chain (Module 5)

- **Scenario Brief**
  Computes top delay-prone airports by month and longest delay chain in route network.
- **Data Structures Used**
  HashMap → airport → month → list of delays
  Graph + Memoized DFS for chain
- **Time and Space Complexity**
  Monthly Aggregation → O(n)
  Longest Chain (DFS) → O(V + E)
- **Sample Input & Output**
  Input: `forced_chains_delay_data.csv`
  Output:

```
Top airports in October: AMD, DEL
Longest delay chain: AMD → CCU → DEL
Total delay: 282 mins
```

- **Code Snippet**

```
double longestDelayFrom(src) {
  if (memo[src]) return memo[src];
  for (neighbor : graph[src])
    memo[src] = max(memo[src], delay + longestDelayFrom(neighbor));
}
```

- **Challenges Faced & How You Solved Them**
  Memoization was key to preventing exponential recursion in multi-hop delay search.

Feature: Gate Allocation System – Advanced Sort (Module 6)

- **Scenario Brief**
  Sorts flights by arrival time, passenger priority, and flight risk category.
- **Data Structures Used**
  Custom Merge Sort on vector<Flight>
  String to Enum mapping for risk/priority
- **Time and Space Complexity**
  Merge Sort → Time: O(n log n), Space: O(n)
- **Sample Input & Output**
  Input: `gate_flights.csv`
  Output:

```
Gate 1 → BA202 | 08:30 | VIP | High
Gate 2 → QR303 | 08:30 | Business | Low
```

- **Code Snippet**

```
if (a.arrivalTime != b.arrivalTime) return a < b;
if (a.priority != b.priority) return a.priority < b.priority;
```

```
return a.risk < b.risk;
```

- **Challenges Faced & How You Solved Them**
  Stable sorting ensured passengers with same time but higher priority came first.

Feature: System Dashboard – Aggregated Ranking (Module 7)

- **Scenario Brief**
  Displays live dashboard of delay-prone routes, baggage jams, and risky connections.
- **Data Structures Used**
  HashMaps for ranking
  Vector sorting for top-k
  Time converters for connection risk
- **Time and Space Complexity**
  Delay + Baggage Ranking → O(n log n)
  Flight Risk Check → O(n)
- **Sample Input & Output**
  Input: delay data + baggage flow
  Output:

```
Top Delays: DEL → MAA
Baggage Jam: Sector 8 (24 bags)
Flight Risk: DEL → BLR (Layover = 15 min)
```

- **Code Snippet**

```
for (pair : baggageSectorCount) sort by descending count;
for (conn : connections) if (layover < 30) mark risky;
```

- **Challenges Faced & How You Solved Them**
  Had to unify multiple datasets from earlier modules and synchronize input parsing.

---

5. Additional Use Case Implementation

Feature: Special Use Case 1 – Baggage Weight Balancer (Extension of Module 3)

- **Scenario Brief**
  Optimally splits a set of bags into two compartments such that the total weights are as equal as possible. Useful in airplane cargo planning for weight distribution.
- **Data Structures Used**
  2D Boolean DP table → `dp[i][j]` where `i = bag count`, `j = cumulative weight`
  Backtracking vector to recover the grouping of bags
- **Time and Space Complexity**
  Let `n = number of bags`, `W = total weight`
- Time: O(n × W)
- Space: O(n × W)

- **Sample Input & Output**
  Input:
  Bags with weights: 32, 20, 28, 24, 18
  Total Weight = 122

  Output:

```
Compartment A (Weight: 62): IDs = 101, 103
Compartment B (Weight: 60): IDs = 102, 104, 105
```

- **Code Snippet**

```
vector<vector<bool>> dp(n+1, vector<bool>(target+1, false));
dp[0][0] = true;
for (int i = 1; i <= n; ++i) {
  for (int w = 0; w <= target; ++w) {
    dp[i][w] = dp[i-1][w] || (w >= weights[i-1] && dp[i-1][w -
weights[i-1]]);
  }
}
```

- **Challenges Faced & Resolution**
  Initially tried greedy sorting by weight, which failed in close-weight scenarios.
  Switched to subset sum DP which guarantees optimal division.

Feature: Special Use Case 2 – Business Route Optimizer (Constrained DFS on Flight Graph)

- **Scenario Brief**
  Finds all valid multi-leg flight paths (≤ 2 hops) from source to destination under business traveler constraints:
- Total cost ≤ $1200
- Max layover ≤ 180 minutes
- At most 2 hops
- **Data Structures Used**
  Flight Graph with extended route timing and cost
  DFS with pruning
  Path vector and results vector
- **Time and Space Complexity**
  Let `b = branching factor`, `k = max depth = 3`
- Time: O(b^k) worst-case, much lower with pruning
- Space: O(p × k) for storing all valid paths
- **Sample Input & Output**
  Input:
  Source = BOM, Destination = MAA
  Output:

```
Valid Routes:
BOM → DEL → MAA
BOM → HYD → MAA
```

```
BOM → MAA
```

- **Code Snippet**

```
void dfs(string curr, int cost, int hops, vector<string>& path) {
  if (hops > 2 || cost > 1200) return;
  if (curr == dest) save path;
  for (auto& next : graph[curr]) {
    if (layover is valid) {
      dfs(next, cost + next.cost, hops + 1, path + next);
    }
  }
}
```

- **Challenges Faced & Resolution**
  Handling time parsing (HH:MM) and comparing layovers correctly was tricky.
  Implemented `parseTime()` and `isLayoverValid()` to encapsulate this logic.
  DFS initially explored all paths — added strict pruning to improve speed.

---

6. Testing & Validation

Category	Details
Number of Functional Test Cases Written	~30 test scenarios across 9 modules using both manual and batch inputs
Edge Cases Handled	- Missing airport codes in routing engine - Invalid baggage data (negative weight, duplicate ID) - No available paths for DFS in special case - Flights with tie-breaker in gate sorting (same time + same priority) - Baggage removal when ID not found in tracker
Known Bugs / Incomplete Features *(if any)*	-No GUI dashboard, all outputs are CLI-based -Layover constraints assume flights are on same day (no cross-day support) -Airport location names from `airports.dat` are not printed (only codes used)

---

7. Final Thoughts & Reflection

• Key Learnings from the Hackathon

- Learned how to balance **algorithmic complexity with modular software design**
- Implemented real-world graph applications using **Dijkstra**, **Bellman-Ford**, and **DFS**
- Practiced advanced data structure integration (BST + Heap + HashMap + DP)

- Applied **constraint-based pruning** and **memoization** effectively

• *Strengths of Your Solution*

- Modular and well-separated architecture (each `.cpp` file maps to one module)
- Code readability and reuse with structured abstractions (`FlightGraph`, `BaggageTracker`, etc.)
- Optimized performance using appropriate DS & algorithms for each task
- Thorough testing and real input data used to simulate actual airport ops

• *Areas for Improvement*

- Could add a **graphical dashboard** using Qt or web tech (HTML/JS)
- Implement persistent caching for delays and route lookups
- Reduce duplicate route paths in special use case outputs
- Add **unit test automation** (e.g., GoogleTest or C++ assert-driven framework)

• *Relevance to Your Career Goals*

- Strongly aligns with my aspiration to work as a **Full Stack ML Developer** or **System Designer**
- Combines both algorithmic depth and system engineering
- Enhanced my ability to turn complex domain problems into well-structured codebases
- Demonstrated use of **DSA in applied problem-solving**, not just theory