**Requirements Specification**

|  |  |
| --- | --- |
| **Client** | Airline |
| **User** | Airplane crew |
| **Functional requirements** | RF\_1. The system must allow uploading passenger information, for a specific flight, from a plain text file.  RF\_2. The system must record the arrival of passengers at the boarding gate.  RF\_3. The system must efficiently search and retrieve complete passenger information when they arrive at the gate.  RF\_4. The system must allow passengers, sorted by the different sections of the aircraft, to board on a first-come, first-served basis.  RF\_5. The system must show to the crew member in charge, the order in which passengers should board the aircraft.  RF\_6. The system must have special rules for first class passengers when boarding, considering data such as: miles accumulated, special attention required, seniors, etc.  RF\_7. The system must establish an order for passengers to leave the plane based on their location, giving priority to those closest to departure, and proximity to the aisle or first served arrival.  RF\_8. The system must show to the crew member in charge, the order in which passengers should leave the aircraft. |
| **Context of the problem** | A recognized airline presents the need for a system whose main objective is to improve the order in the process of entry and exit of the aircraft. The system is required to show the crew member in charge in what order passengers must enter and exit.  Main goal is to reward the punctuality of passengers by entering the plane on a first-come, first-served basis. However, the airline wants the first class to have some special rules when they are called for entry to the plane, that is, not only considering their order of arrival, but also prioritizing other data. |
| **Non-functional requirements** | RNF\_1. The system must have the capacity to handle large amounts of passenger data. |

**Requirements analysis table**

|  |  |  |  |
| --- | --- | --- | --- |
| Name or identifier | **R1:** Upload passenger information | | |
| Summary | The system must allow uploading passenger information, for a specific flight, from a plain text file. | | |
| Inputs | **Entry name** | **Data type** | **Selection or repetition condition** |
| file | String | It must be indicate a valid URL where the text file comes from. |
| flight \_id | String | It must be an existing flight id. |
| General activities required to achieve results | When the flight id is found, the following is needed:  To receive the file.  To decerealize the information, located in lines, in passenger type objects. | | |
| Result or Postcondition | Information converted into Passengers | | |
| Outputs | **Output name** | **Data type** | **Selection or repetition condition** |
| message | String | A message where contains the number of passengers uploaded in the flight. |
|  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Name or identifier | **R2:** Record arrival passenger. | | |
| Summary | The system must record the arrival of passengers at the boarding gate. | | |
| Inputs | **Entry name** | **Data type** | **Selection or repetition condition** |
| passenger | passenger | It must validate the information and the existing passenger. |
|  |  |  |
| General activities required to achieve results | Stack the passenger in a priority queue, so that he/she can be removed in time of arrival. | | |
| Result or Postcondition | A recordered passenger | | |
| Outputs | **Output name** | **Data type** | **Selection or repetition condition** |
| message | String | A message where it says that the passenger was recordered. |
|  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Name or identifier | **R3:** Search passenger | | |
| Summary | The system must efficiently search and retrieve complete passenger information when they arrive at the gate. | | |
| Inputs | **Entry name** | **Data type** | **Selection or repetition condition** |
| id | String |  |
|  |  |  |
| General activities required to achieve results | Search the passenger in the flight list and obtain the number of the arrival list. | | |
| Result or Postcondition | The passenger validated | | |
| Outputs | **Output name** | **Data type** | **Selection or repetition condition** |
| message | String | A message where contains the basic passenger info and the orden to seat. |
|  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Name or identifier | **R4:** Allow passenger to seat | | |
| Summary | The system must allow passengers, sorted by the different sections of the aircraft, to board on a first-come, first-served basis. | | |
| Inputs | **Entry name** | **Data type** | **Selection or repetition condition** |
| id | String |  |
|  |  |  |
| General activities required to achieve results | Obtain the number of the asigned number seat accordig to the ID passenger. | | |
| Result or Postcondition | The passenger validated | | |
| Outputs | **Output name** | **Data type** | **Selection or repetition condition** |
| message | String | It’s given to a passenger its number seat. |
|  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Name or identifier | **R5:** View order to get in by crew member in charge | | |
| Summary | The system must show to the crew member in charge, the order in which passengers should board the aircraft. | | |
| Inputs | **Entry name** | **Data type** | **Selection or repetition condition** |
| Flight\_id | String | It must be an existent id |
|  |  |  |
| General activities required to achieve results | The user has to be authorized.  When it’s found the id of the flight, it must show the list of passengers according to arrival time (first to arrive until last to arrive). | | |
| Result or Postcondition | The passenger validated | | |
| Outputs | **Output name** | **Data type** | **Selection or repetition condition** |
| message | String | A message where contains the arrival time list passengers. |
|  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Name or identifier | **R6:** First class to attend | | |
| Summary | The system must have special rules for first class passengers when boarding, considering data such as: miles accumulated, special attention required, seniors, etc. | | |
| Inputs | **Entry name** | **Data type** | **Selection or repetition condition** |
| passenger | passenger | If age is more than 65 years old  If miles accumulated is true  If special attention is true |
|  |  |  |
| General activities required to achieve results | Set in the arrival early list to people with those condition. | | |
| Result or Postcondition | The passenger validated | | |
| Outputs | **Output name** | **Data type** | **Selection or repetition condition** |
| message | String | A message where contains the rules for first class people. |
|  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Name or identifier | **R7:** Order to leave | | |
| Summary | The system must establish an order for passengers to leave the plane based on their location, giving priority to those closest to departure, and proximity to the aisle or first served arrival. | | |
| Inputs | **Entry name** | **Data type** | **Selection or repetition condition** |
| id | String |  |
|  |  |  |
| General activities required to achieve results | Search the passenger in the flight list and obtain the number of the arrival list. | | |
| Result or Postcondition | The passenger validated | | |
| Outputs | **Output name** | **Data type** | **Selection or repetition condition** |
| message | String | A message where contains the basic passenger info and the orden to seat. |
|  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Name or identifier | **R8:** View order to leave by crew member in charge | | |
| Summary | The system must show to the crew member in charge, the order in which passengers should leave the aircraft. | | |
| Inputs | **Entry name** | **Data type** | **Selection or repetition condition** |
| id | String |  |
|  |  |  |
| General activities required to achieve results | Search the passenger in the flight list and obtain the number of the arrival list. | | |
| Result or Postcondition | The passenger validated | | |
| Outputs | **Output name** | **Data type** | **Selection or repetition condition** |
| message | String | A message where contains the basic passenger info and the orden to seat. |
|  |  |  |

**Engineering method**

**1. Problem identification**

**Needs and symptoms:**

* The airline does not have software that allows the crew to maintain order when boarding and descending the plane.
* The airline wants to improve the order in which the plane enters and leaves by considering the punctuality of passengers.
* Special priority rules are needed for first class passengers.
* Passenger data needs to be uploaded to the system.
* The system must have the greatest possible efficiency in the search and obtaining of information, since the amount of data it will use will be very large.

**Problem definition:**

The airline requires software that tells the crew in charge:

* The order in which passengers should enter the aircraft based on arrival time or other parameters.
* The order in which passengers must exit the aircraft based on their proximity to departure or first served arrival.

**2. Collection of Information**

Information, to be loaded, of the passengers: Identity document, nationality, date of birth, seat information (Type, section, number, etc.).

Boarding priority hierarchy:

1. Special attention: passengers who require specific attention or care, such as those with reduced mobility or special medical needs.
2. Seniors: Passengers who qualify as seniors.
3. Travel frequency: Passengers sorted by the number of miles accumulated, frequency and number of trips.

Search algorithms:

* Sequential search: It consists of comparing the item that is searched with each item in the list until it is found.
* Binary search: It is the most efficient algorithm for searching within a list of ordered items. It is based on repeatedly dividing the list in half, taking half that the element can contain and repeating the process until reducing the possible locations to just one and finding the item.

Sorting algorithms:

* Bubble sorting: Adjacent elements are compared, and we push the lightest values up the list (the heaviest, or "largest", are left down).
* Push sorting: You have two sets of data, one ordered and one unordered. The idea is to iterate the unordered set by inserting each element in the correct position within the ordered set.
* Quick sort: It is called the fastest sorting algorithm known. It works by selecting an item from the list to be referred to as a "pivot." Then, the minor elements are moved to the pivot, to the left of it, and the major ones to the right. This process is repeated recursively with the obtained sublists (left and right of the pivot).

**3. Search for creative solutions**

**Solution to the problem of priority of entry and exit:**

Solution 1.

Use priority queues next to mounds to manage the order of passengers.

Solution 2.

Use matrices for the management of seat information.

**Solution to the storage and search of passenger data:**

Alternative 1.

Use a list or array for the main saving of passenger data and perform the management and search of data using the methods: "Quick sort" as the main sorting algorithm (to ensure that the data entered is sorted) and "binary search" to perform the search within the datasets.

Alternative 2.

Use a Binary Search Tree for the main saving of passenger data and perform the search quickly within the tree.

Alternative 3.

Use a hash table to save all the passenger’s data. They will be stored by their id’s, making it easier to search for them later.

**4. Transition from ideas to main designs**

**Review of ideas:**

* **Entry priority:**
  + Alternative 1 – Different queues for different sections and classes:
    - Does not offer too much flexibility with passengers’ priority levels.
    - It makes easy/comprehensible to handle the data.
  + Alternative 2 – Priority heap:
    - It offers the possibility to assign different priorities to passengers considering multiple parameters.
    - It provides speed in the extraction of data for the moment of call to entry of the passengers.
    - It needs to calculate each passenger priority and put it in its respective index.
* **Exit priority:**
  + Alternative 1 – Matrix:
    - It facilitates the emulation and view of the space of the plane (the seats).
    - It facilitates the exit order, being as simple as going through the entire matrix of passengers.
  + Alternative 2 – Priority heap:
    - It offers the possibility to assign different priorities to passengers considering multiple parameters.
    - It provides speed in the extraction of data for the moment of call to entry of the passengers.
    - It needs to calculate each passenger priority and put it in its respective index.
* **Storage and search:**
  + Alternative 1 – Ordered arrangement and binary search:
    - It implies a static context of the data.
    - It would involve the declaration of empty spaces (sufficient to ensure passenger storage capacity)
    - Search problems due to poor order
  + Alternative 2 – Binary search tree:
    - It provides dynamism to data management.
    - It provides a higher speed in the deletion of data within the tree.
    - If it isn’t an auto balanced tree, it might be highly inefficient.
    - If it is an auto balanced tree, it would be needing to continuously rotate its nodes.
  + Alternative 3 – Hash table:
    - It provides dynamism to data management due to it has no limit of objects.
    - The input and output of data is almost constant because it is based on a set of unique keys for each element.

**5. Evaluation and selection of solutions**

**Criteria:**

* Criteria A – Data input and output. The algorithm’s efficiency is:
  + [4] Constant
  + [3] Almost constant
  + [2] Logarithmical
  + [1] Lineal
* Criteria B – Flexibility. The algorithm allows:
  + [4] To differentiate between multiple parameters and specify a priority level/hierarchy.
  + [3] To differentiate between multiple parameters but not to specify a priority level/hierarchy.
  + [2] To barely differentiate between specific parameters.
  + [1] Does not provides any flexibility.
* Criteria C – Dynamism:
  + [2] Dynamic context
  + [1] Static context

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Problem** | **Alternative** | **Criteria A**  **40%** | **Criteria B**  **30%** | **Criteria C**  **30%** | **Total** |
| **Entry priority** | Alternative 1 – queues | [3] | [2] | [2] | 2.4 |
| Alternative 2 - Priority heap | [2] | [4] | [2] | 2.6 |
| **Exit priority** | Alternative 1 – matrix | [4] | [4] | [1] | 3.1 |
| Alternative 2 - Priority heap | [2] | [4] | [2] | 2.6 |
| **Data storage** | Alternative 1 - Arrays | [2] | Does not apply | [1] | 1.1 |
| Alternative 2 – Binary search tree | [2] | [2] | 1.4 |
| Alternative 3 - Hash table | [3] | [2] | 1.8 |

Based on the scores for each alternative, the best solutions, because they got the higher ones, would be:

**Entry priority**: Alternative 2 – Priority heap.

**Exit priority**: Alternative 1 – Matrix.

**Data storage**: Alternative 3 – Hash table.