**Functional Requirements**

|  |  |
| --- | --- |
| **Name** | R.#1 Register restaurant |
| **Summary** | Allows the user to register a restaurant given its data. |
| **Input** | |
| Name, NIT, administrator’s name. | |
| **Output** | |
| The restaurant will have been registered successfully. | |

|  |  |
| --- | --- |
| **Name** | R.#2 Register product |
| **Summary** | Allows the user to register a product given its data. |
| **Input** | |
| ID, name, description, price, NIT, restaurant | |
| **Output** | |
| The product will have been registered successfully. | |

|  |  |
| --- | --- |
| **Name** | R.#3 Register client |
| **Summary** | Allows the user to register a client given their data. The client will be inserted sorted by last and then first name. |
| **Input** | |
| ID type, ID number, full name, phone number, address. | |
| **Output** | |
| The client will have been registered successfully. | |

|  |  |
| --- | --- |
| **Name** | R.#4 Register order |
| **Summary** | Allows the user to place an order given its specifications. |
| **Input** | |
| Client’s ID, restaurant’s NIT, each product’s code and amount. | |
| **Output** | |
| The order will have been registered successfully. | |

|  |  |
| --- | --- |
| **Name** | R.#5 Update restaurant’s data |
| **Summary** | Allows the user to change certain values for a specific restaurant given their NIT. |
| **Input** | |
| Current NIT, new NIT, name, administrator’s name. | |
| **Output** | |
| The restaurant’s data will have been successfully updated. | |

|  |  |
| --- | --- |
| **Name** | R.#6 Update client’s data |
| **Summary** | Allows the user to change certain values for a specific client given their ID. |
| **Input** | |
| Current ID, new ID, full name, phone number, address. | |
| **Output** | |
| The client’s data will have been updated successfully. | |

|  |  |
| --- | --- |
| **Name** | R.#7 Update order’s data |
| **Summary** | Allows the user to change certain values for a specific order given its ID. |
| **Input** | |
| ID, whether date is going to be updated, Client’s ID, restaurant’s NIT, each product’s code, and amount. | |
| **Output** | |
| The order’s data will have been updated. | |

|  |  |
| --- | --- |
| **Name** | R.#8 Update order’s status |
| **Summary** | Allows the user to advance the order’s state forward if it isn’t already in the maximum state. |
| **Input** | |
| New status. | |
| **Output** | |
| The order’s status will be updated. | |

|  |  |
| --- | --- |
| **Name** | R.#9 Save data seamlessly |
| **Summary** | The program’s data will be serialized every time it is updated. |
| **Input** | |
| None. | |
| **Output** | |
| The program’s data will be serialized in a default file. | |

|  |  |
| --- | --- |
| **Name** | R.#10 Export orders’ data |
| **Summary** | The orders’ data is exported into a csv file and is sorted by the following criteria: ascending restaurant NIT, descending client ID, ascending order date and ascending product ID. |
| **Input** | |
| File name, CSV Separator. | |
| **Output** | |
| All order data will be exported into a csv file. | |

|  |  |
| --- | --- |
| **Name** | R.#11 List restaurants |
| **Summary** | Restaurants are shown in alphabetical order. |
| **Input** | |
| None. | |
| **Output** | |
| Restaurants will be displayed. | |

|  |  |
| --- | --- |
| **Name** | R.#12 List Clients |
| **Summary** | Clients are shown ordered by descending phone number |
| **Input** | |
| None. | |
| **Output** | |
| Clients will be displayed. | |

|  |  |
| --- | --- |
| **Name** | R.#13 Search client by name |
| **Summary** | Searches for a client given their name and displays the search time. |
| **Input** | |
| Name. | |
| **Output** | |
| Client and search time. | |

|  |  |
| --- | --- |
| **Name** | R.#14 Import data |
| **Summary** | Imports object data from csv (The first line is not read as it states what each column is) |
| **Input** | |
| Data type, CSV File. | |
| **Output** | |
| Data will be imported. | |

**Non-functional requirements**

* The user can only update the status of an order forwards.
* The search function must be efficient
* The program must use at least two of the sorting algorithms bubble, selection and insertion.
* The program must do at least one sort using the Comparable interface and one using the Comparator interface, using the Collections or Arrays sort in both cases.
* Test must use a CSV file with 1000 entries.
* All clients must have exactly two names separated by a space.