**Restaurant Recommendation System**

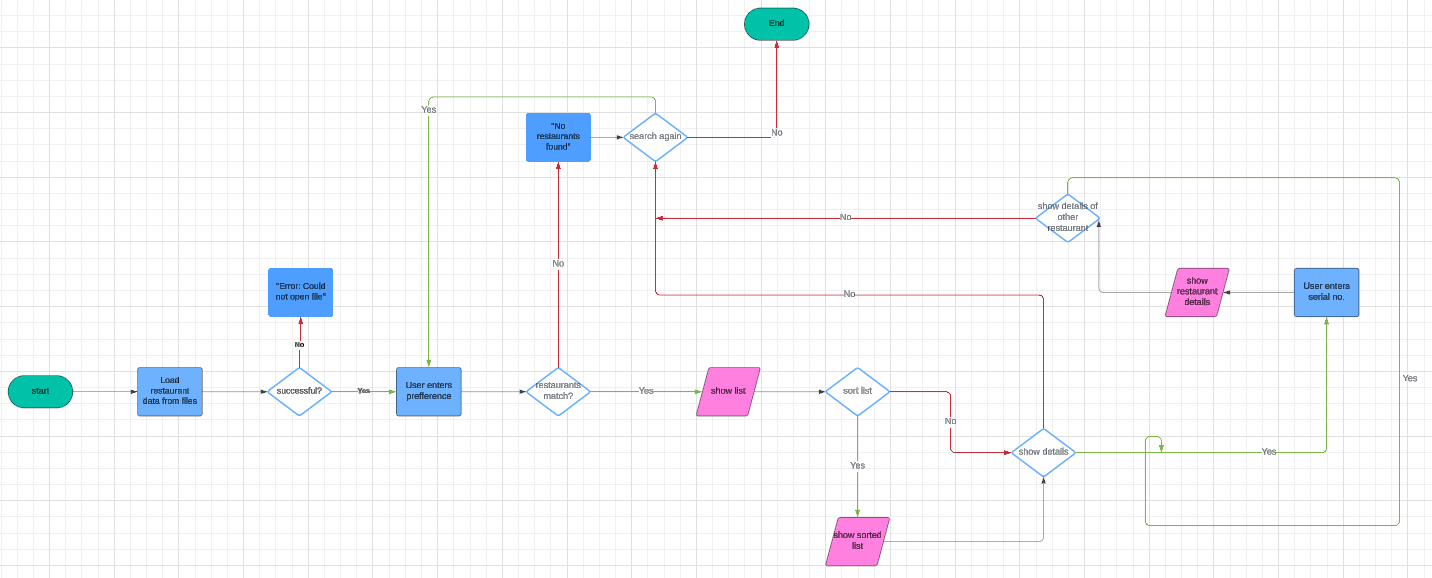
**Milestone 2**

Saad Thaplawala

**Tools and Libraries:**

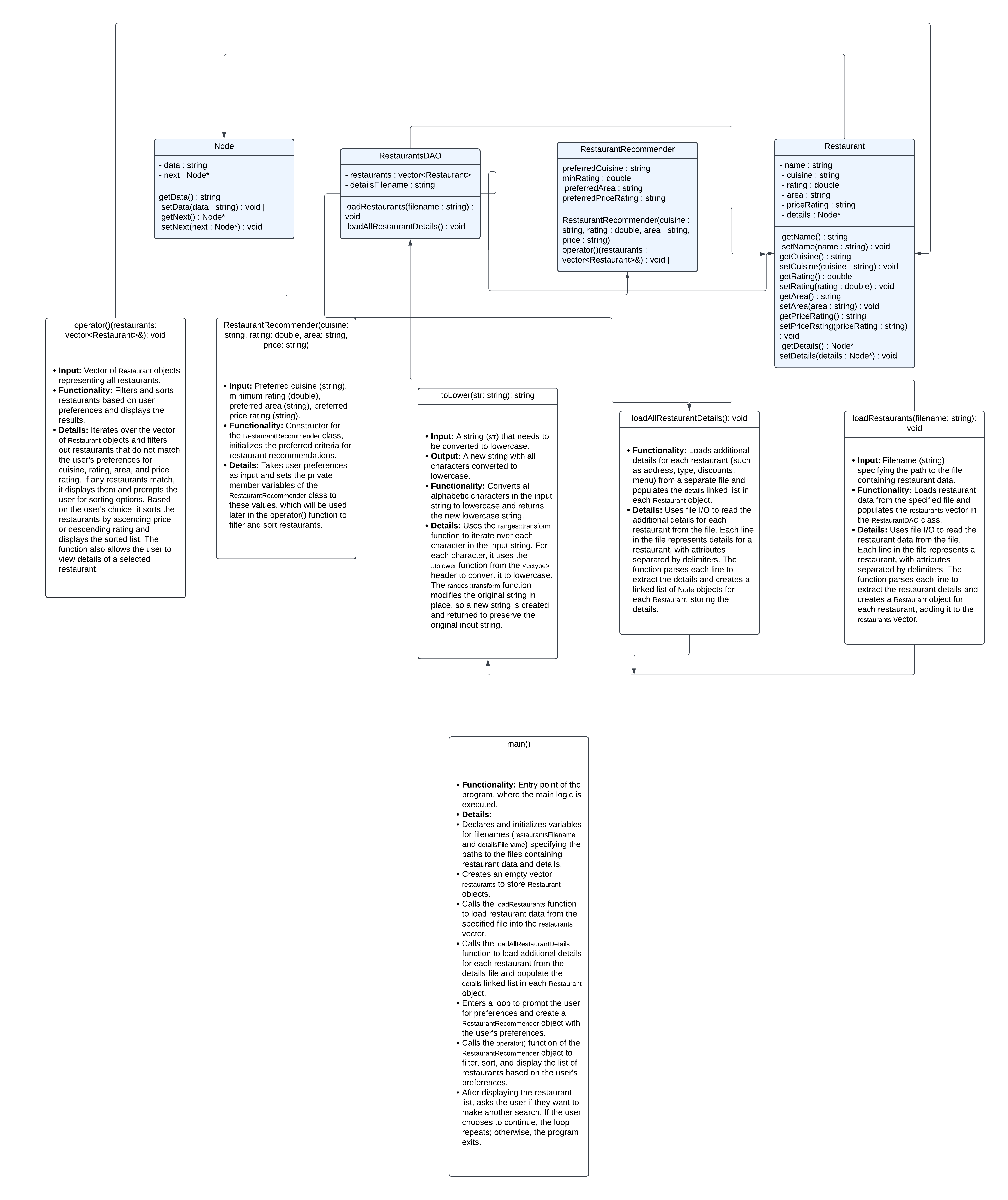
* IDE(Integrated Development Environment):
  + Clion.
* Debugging Tool:
  + Clion code debugger.

Flow chart:



Class diagram:

<https://lucid.app/lucidchart/addf23a1-ffa0-40ca-9bc4-1220d344a71c/edit?invitationId=inv_61a648cd-9b3e-4909-bbb0-8284de928e5c>



 Input **Validation Test Cases:**

* Test for valid input within specified ranges:
  + Input: Valid cuisine (e.g., "Italian"), valid rating (e.g., 4.5), valid area (e.g., "Downtown"), valid price rating (e.g., "$$").
  + Expected Output: List of restaurants matching the criteria.
* Test for invalid input (e.g., negative numbers, non-numeric characters):
  + Input: Negative rating (-1), non-numeric characters for rating (e.g., "abc").
  + Expected Output: Error message indicating invalid input.

 Boundary **Test Cases:**

* Test for empty cuisine:
  + Input: Empty string for cuisine.
  + Expected Output: List of restaurants with any cuisine.
* Test for minimum valid rating:
  + Input: Minimum valid rating (0).
  + Expected Output: List of restaurants with ratings greater than or equal to 0.

 **Edge Cases Test Cases:**

* Test for maximum valid rating:
  + Input: Maximum valid rating (e.g., 5).
  + Expected Output: List of restaurants with ratings less than or equal to 5.
* Test for maximum price rating:
  + Input: Maximum price rating (e.g., "$$$$").
  + Expected Output: List of restaurants with price rating equal to "$$$$".

 **Sorting Test Cases:**

* Test for sorting by ascending price:
  + Input: Sort by ascending price.
  + Expected Output: List of restaurants sorted by ascending price.
* Test for sorting by descending rating:
  + Input: Sort by descending rating.
  + Expected Output: List of restaurants sorted by descending rating.

 **Details Display Test Cases:**

* Test for displaying details of a selected restaurant:
  + Input: Select a restaurant from the list.
  + Expected Output: Display the details of the selected restaurant (e.g., address, type, discounts, menu).

 **No Match Test Cases:**

* Test for no restaurants matching the criteria:
  + Input: Criteria that do not match any restaurants (e.g., cuisine "desi").
  + Expected Output: Message indicating no restaurants found matching the criteria.

 **Multiple Matches Test Cases:**

* Test for multiple restaurants matching the criteria:
  + Input: Criteria that match multiple restaurants (e.g., cuisine "Mexican" in an area with multiple Mexican restaurants).
  + Expected Output: List of all restaurants matching the criteria.

 **Repeated Searches Test Cases:**

* Test for performing multiple searches:
  + Input: Perform multiple searches with different criteria.
  + Expected Output: Correctly display the results for each search without interference from previous searches.

 **Functionality Test Cases:**

* Test restaurant filtering based on cuisine:
  + Input: Specify a cuisine and check if only restaurants of that cuisine are listed.
* Test restaurant filtering based on rating:
  + Input: Specify a minimum rating and check if only restaurants with equal or higher ratings are listed.
* Test restaurant filtering based on area:
  + Input: Specify an area and check if only restaurants in that area are listed.
* Test restaurant filtering based on price rating:
  + Input: Specify a price rating and check if only restaurants with that price rating are listed.
* Test sorting by ascending price:
  + Input: Sort the list by ascending price and check if restaurants are listed in the correct order.
* Test sorting by descending rating:
  + Input: Sort the list by descending rating and check if restaurants are listed in the correct order.
* Test displaying restaurant details:
  + Input: Select a restaurant and check if its details are displayed correctly.

 **Edge Case Test Cases:**

* Test for an empty list of restaurants:
  + Input: Load an empty list of restaurants and check if the program handles it gracefully.
* Test for maximum valid rating:
  + Input: Specify the maximum valid rating (e.g., 5) and check if the program handles it correctly.
* Test for maximum price rating:
  + Input: Specify the maximum price rating (e.g., "$$$$") and check if the program handles it correctly.

 **Error Handling Test Cases:**

* Test for invalid file paths:
  + Input: Provide an invalid file path for loading restaurant data and check if the program handles the error gracefully.
* Test for invalid user input:
  + Input: Provide invalid input (e.g., non-numeric characters for rating) and check if the program displays an appropriate error message.
* Test for out-of-range values:
  + Input: Provide out-of-range values (e.g., negative rating) and check if the program handles them correctly.
* Test for empty input:
  + Input: Provide empty input for cuisine, area, and price rating and check if the program handles it gracefully.
* Case insensitive input:
  + Input: Provide input e.g. fast food or Fast Food both should work.

 **Combination Test Cases:**

* Test combinations of features:
  + Input: Specify a cuisine, minimum rating, area, and price rating, and check if the program filters and sorts the restaurants correctly based on all criteria.