**CHEETOS MANAGEMENT SYSTEM**



Session: 2022 – 2025

**Submitted by:**

Taha Shahid 2022-CS-197

**Supervised by:**

Dr. Awais Hassan

Department of Computer Science

**University of Engineering and Technology**

Contents

**[ABSTRACT](#_Toc128704438)** [3](#_Toc128704438)

[**SHORT DESCRIPTION OF CHEETOS MANAGEMENT SYSTEM** 4](#_Toc128704439)

[**USERS OF APPLICATION** 4](#_Toc128704440)

[**FUNCTIONAL REQUIREMENTS** 4](#_Toc128704441)

[ **ADMIN** 4](#_Toc128704442)

[ **EMPLOYEE** 6](#_Toc128704443)

[**Wireframes** 7](#_Toc128704444)

[**Figure 1: Initial Sign Up** 7](#_Toc128704445)

[**Figure 2: Sign in screen** 8](#_Toc128704446)

[**Figure 3: Admin main menu** 9](#_Toc128704447)

[**Figure 4: Adding ingredients to recipe table** 9](#_Toc128704448)

[**Figure 10: Viewing flavors added in the system** 12](#_Toc128704449)

[**Figure 11: Adding manufacturing orders** 13](#_Toc128704450)

[**Figure 12: Updating manufacturing order** 13](#_Toc128704451)

[**Figure 13: Delete manufacturing order** 13](#_Toc128704452)

[**Figure 16: Updating user** 15](#_Toc128704453)

[**Figure 17: Deleting user** 15](#_Toc128704454)

[**Figure 18: Viewing users** 16](#_Toc128704455)

[**Figure 22: Employee deleting request for ingredients** 18](#_Toc128704456)

[**Figure 25: Employee viewing ingredients received** 19](#_Toc128704457)

[**Figure 29: Employee requesting an addition in the recipe table** 21](#_Toc128704458)

[**Figure 30: Employee logging extra ingredients** 22](#_Toc128704459)

[**Data Structures (Parallel Arrays)** 22](#_Toc128704460)

[**Function Prototypes** 23](#_Toc128704461)

[**Functions Working Flow** 24](#_Toc128704462)

[**Complete Code of the Business Application** 24](#_Toc128704463)

# **ABSTRACT**

Cheetos Management System is an application that allows easier management of admins, employees, orders, and requests, in a Cheetos manufacturing factory. The applications helps in solving numerous problems that can arise inside a Cheetos factory, including, but not limited to, communication issues, order management issues, etcetera. This system solves these issues by dividing them into numerous smaller and easily solvable sub-problems. After that, through the use of many independent functions, each serving its own purpose in solving all the sub-problems, the software applications provides an easy interface for a more efficient management of the factory. A major problem that can occur inside the factory is that updating data in one portion of the factory might not automatically update data in other portions. However, another problem that occurred using parallel arrays, was that particular groups of data were distributed throughout the program. Where others looked at this as a problem, this project makes use of such distribution in an advantageous manner. By sharing particular data groups, it can be ensured that all portions of the program remain updated. While such a system might be undesirable for some systems, the Cheetos management system tries to use this manner of distribution to its advantage.

# **SHORT DESCRIPTION OF CHEETOS MANAGEMENT SYSTEM**

An initial sign up functionality is provided by the Cheetos management system. This function basically uses file handling to determine whether the application has been run on a particular system before or not. If not, a one-time-only sign up prompt is presented to sign up the first admin of the system. Once registered, the system has safeguards to make sure that at least one admin remains in the user arrays. Employees and admins cannot just sign up their accounts because that would violate the security of the system. Only an admin can add, edit and delete another employee account.

# **USERS OF APPLICATION**

* Admin (responsible for management of the factory)
* Employee (workers in the Cheetos management factory)

# **FUNCTIONAL REQUIREMENTS**

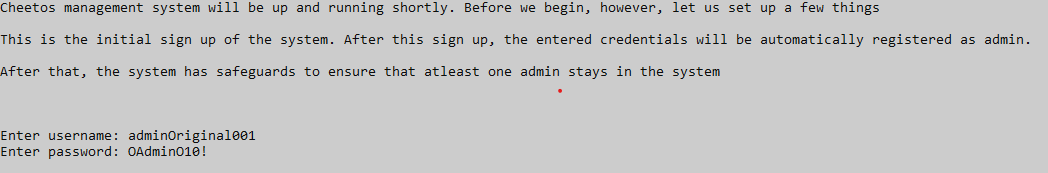
## **ADMIN**

|  |  |  |  |
| --- | --- | --- | --- |
|  | *As a* | *I want to perform* | *So that I can* |
|  | Admin | Create an ingredients order | Prepare necessary ingredients for the manufacturing of Cheetos |
|  | Admin | View an ingredient order | View ingredients orders in detail |
|  | Admin | Update an ingredient order | Keep orders up to date |
|  | Admin | Delete an ingredient order | To avoid ordering unnecessary ingredients |
|  | Admin | Create a manufacturing order | To tell employees their orders |
|  | Admin | View a manufacturing order | To view what products are currently being manufactured in the factory |
|  | Admin | Update a manufacturing order | To keep manufacturing up to date |
|  | Admin | Delete a manufacturing order | To save employees the trouble of completing orders that are no longer necessary |
|  | Admin | View recommended ingredients | To view which ingredients have been requested by the employees and which ingredients were in shortage |
|  | Admin | View excess ingredients | To view which ingredients’ quantities were miscalculated so that admins and engineers can figure out why the miscalculation happened |
|  | Admin | Add ingredients to the recipe table | Ensure that the ingredients ordered belong to the table and no unnecessary ingredients have been ordered |
|  | Admin | View ingredients in the recipe table | See if the recipe table needs updating before ordering some new ingredient to be used in manufacturing |
|  | Admin | Delete ingredients from the table | Keep recipes of the company up to date |
|  | Admin | View employees’ suggestions | Get ideas on making the company better |
|  | Admin | Add user | Add employees and other admins in the system |
|  | Admin | View user info | check user info before determining If they need updating |
|  | Admin | Update user info | Update info due to security concerns or update(promote) a user’s role from employee to admin |
|  | Admin | Delete user info | Make room for other users as user capacity of the factory is limited |
|  | Admin | Add flavors in the system | Ensure that only the flavors that have been officially added in the system, are being manufactured |
|  | Admin | View flavors in the system | View flavors in system before determining whether to add, update or delete a flavor |
|  | Admin | Delete flavors from the system | Delete flavors that are no longer marketable |

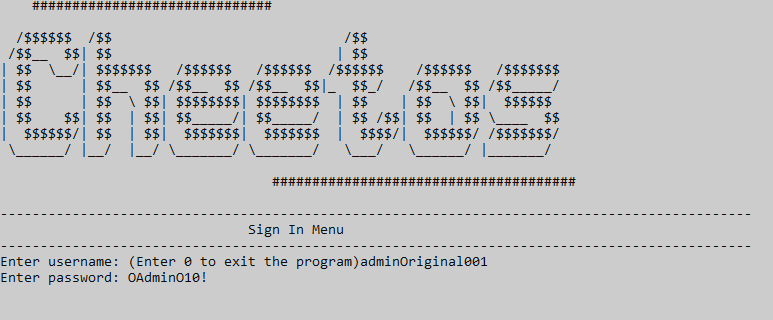
## **EMPLOYEE**

|  |  |  |
| --- | --- | --- |
| Employee | Upload request for ingredients | Ask for ingredients that I think we’ll need |
| Employee | View ingredient requests | Ensure the authenticity of requests |
| Employee | Edit ingredient requests | To make requests up-to-date before they are considered by the admins |
| Employee | Delete ingredient requests | Delete requests that are no longer necessary |
| Employee | View current orders | See what orders need manufacturing |
| Employee | View ingredients received | See what ingredients are available for use |
| Employee | Log shortage of ingredients | Inform admins of a potential glitch in the system |
| Employee | Give a suggestion | Inform admins of potential strategies to improve the factory |
| Employee | View suggestions | Consider my own suggestions and determine if they can be improved |
| Employee | Request for editing in recipes | Add new ingredients in the table to be able to order them in my requests |
| Employee | View ingredients recipe table | See which ingredients can be ordered through factory |
| Employee | View flavors add in the system | See which flavors are the company supposed to manufacture |
| Employee | Log excess of ingredients | Admins can handle cost |

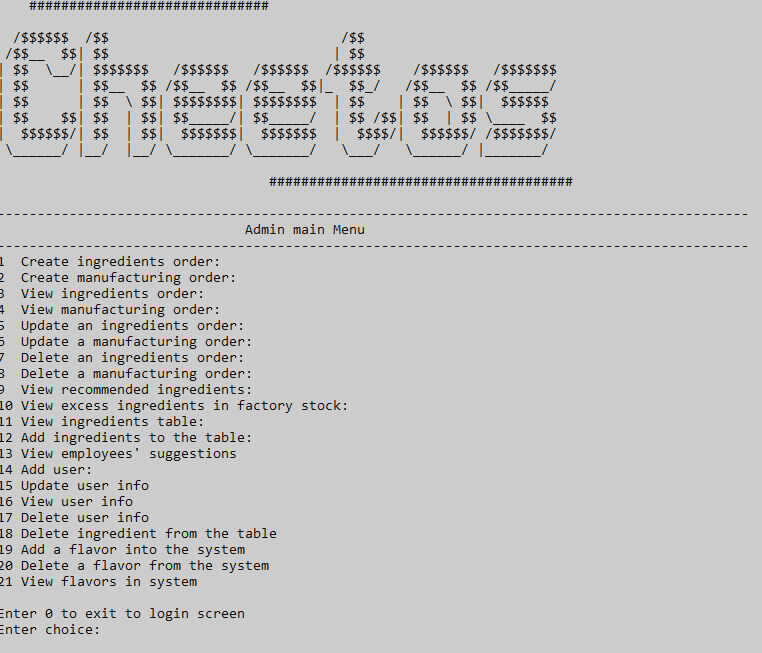
# **Wireframes**

****

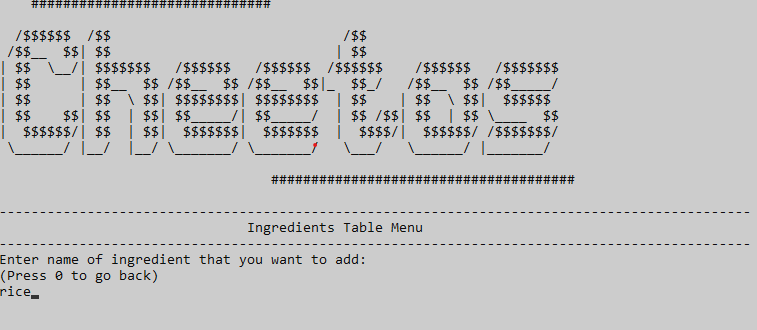
## **Figure 1: Initial Sign Up**

****

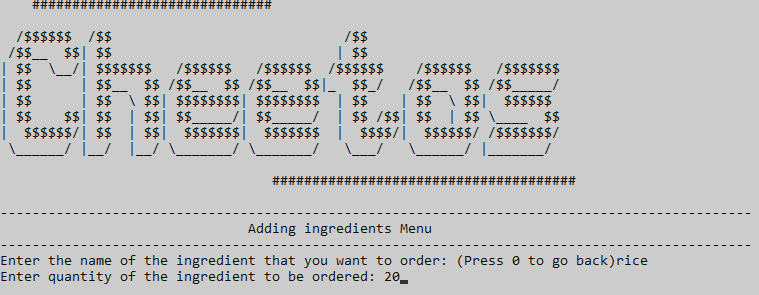
## **Figure 2: Sign in screen**

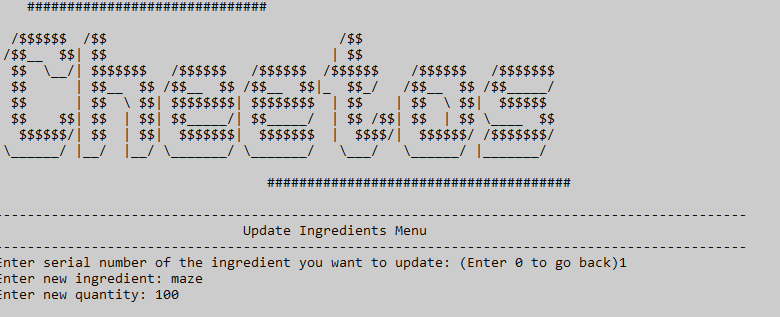


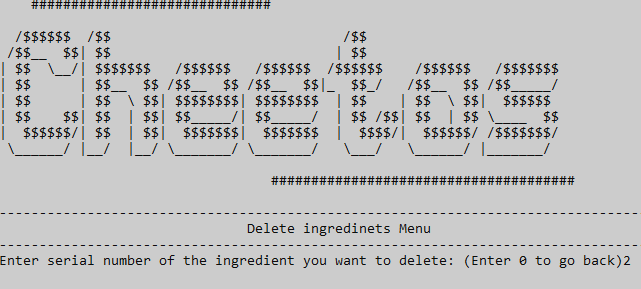
## **Figure 3: Admin main menu**

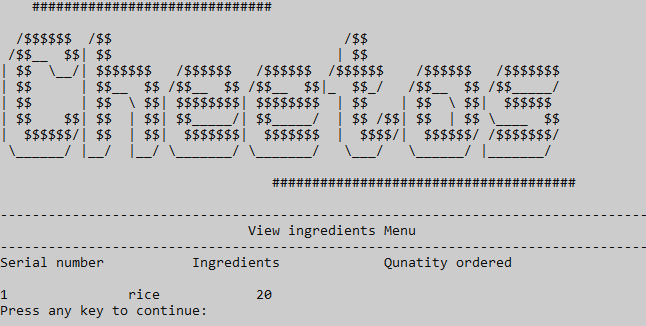


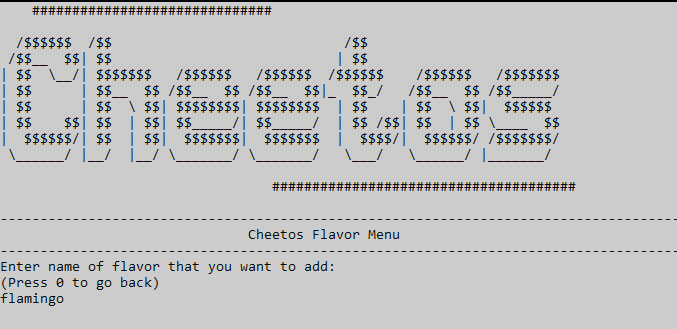
## **Figure 4: Adding ingredients to recipe table**

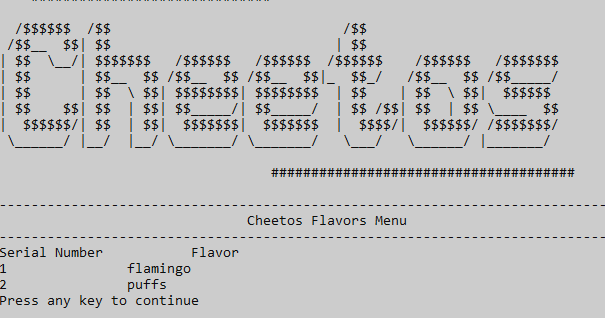
**Figure 5: Adding ingredients order**

**Figure 6: Updating ingredients order**

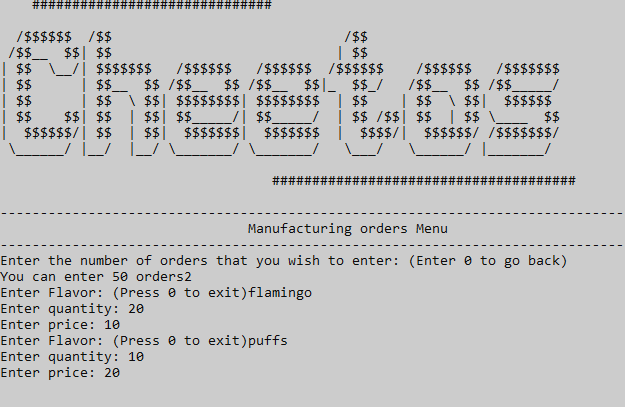
**Figure 7: Deleting ingredients order**

**Figure 8: View ingredients orders**

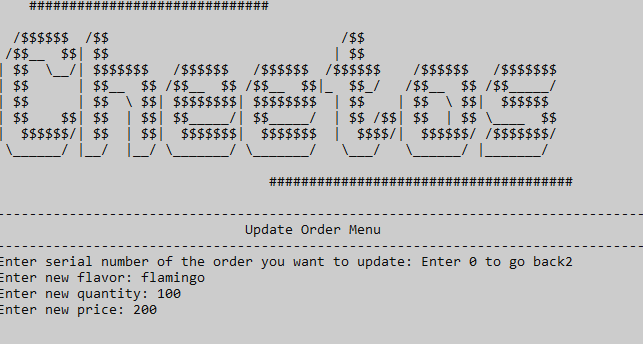
**Figure 9: Adding flavors in the system**



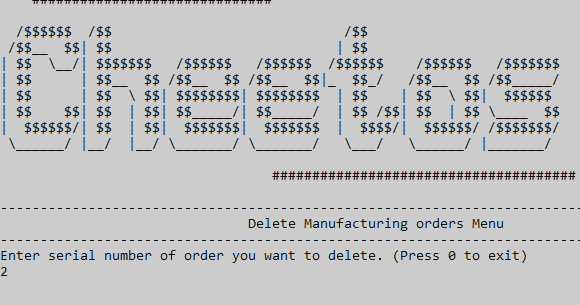
## **Figure 10: Viewing flavors added in the system**



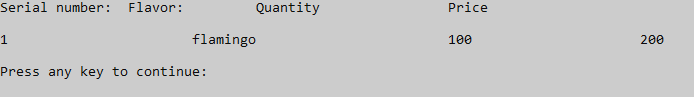
## **Figure 11: Adding manufacturing orders**

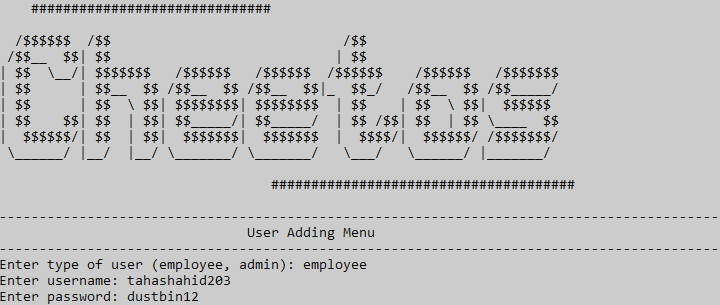


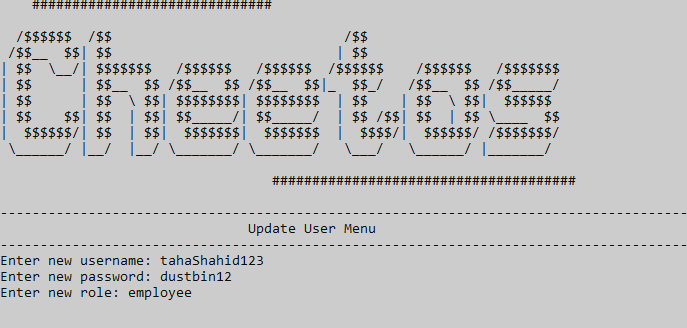
## **Figure 12: Updating manufacturing order**



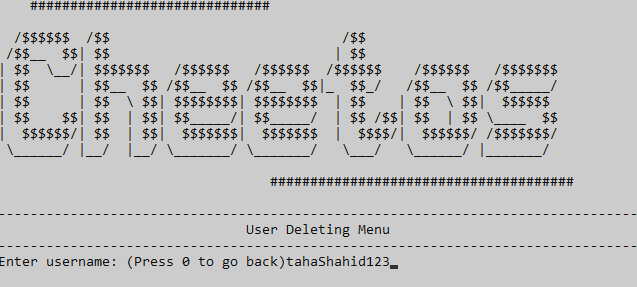
## **Figure 13: Delete manufacturing order**

**Figure 14: Viewing manufacturing orders**

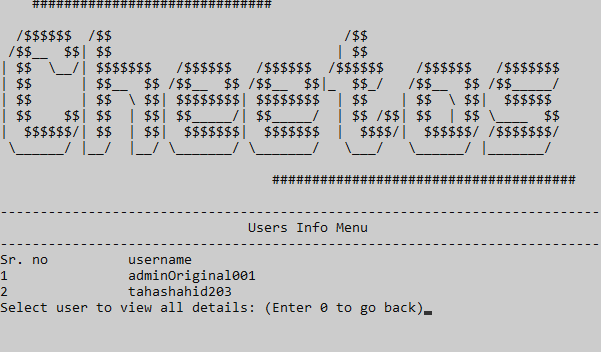
**Figure 15: User adding menu**



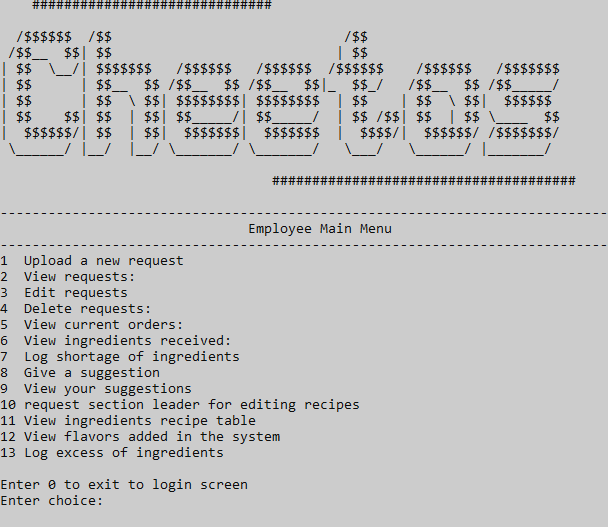
## **Figure 16: Updating user**

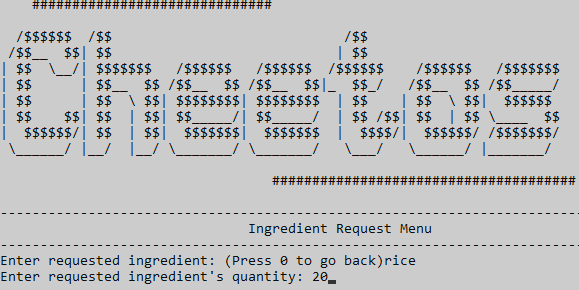


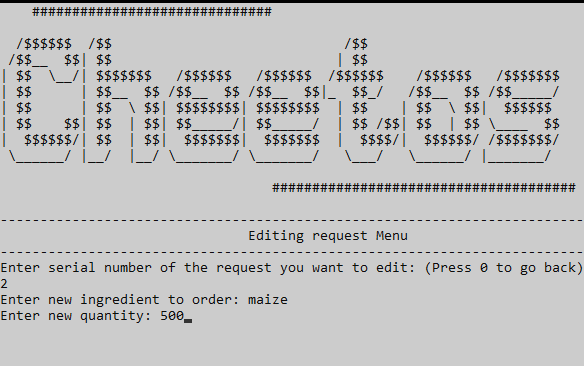
## **Figure 17: Deleting user**

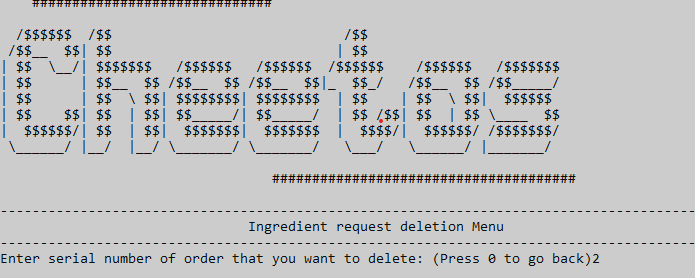


## **Figure 18: Viewing users**

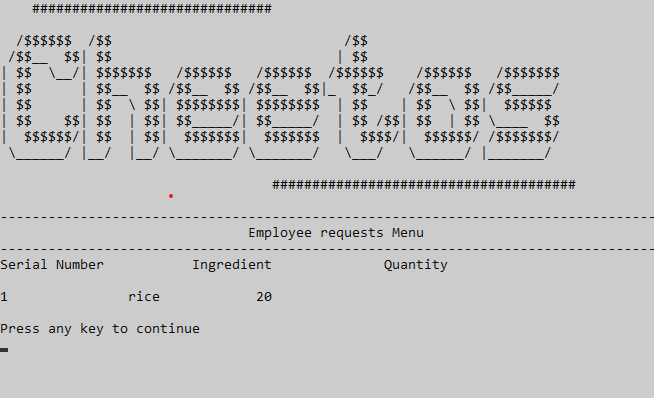
**Figure 19: Employee main menu**

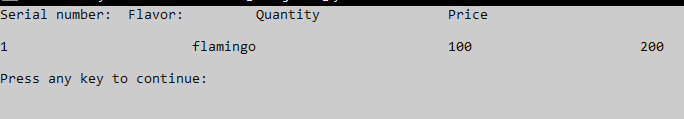
**Figure 20: Employee requesting for ingredients**

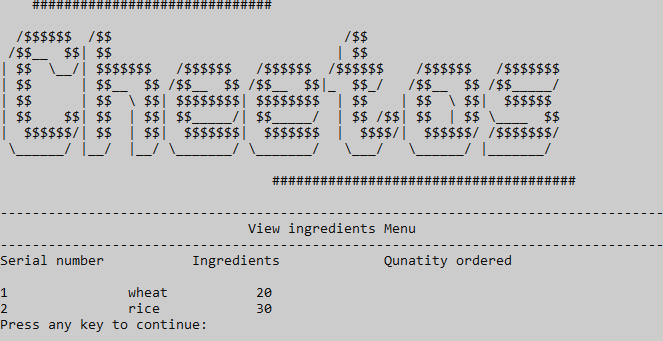
**Figure 21: Employee editing request for ingredients**



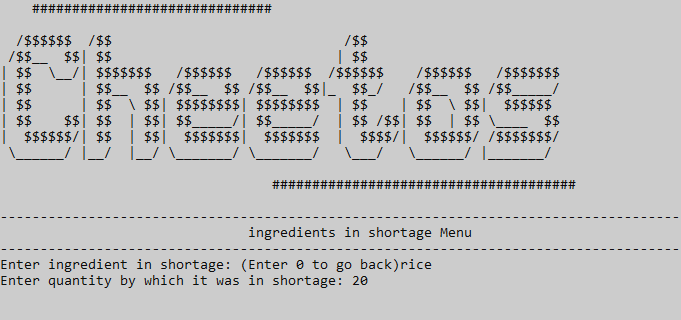
## **Figure 22: Employee deleting request for ingredients**

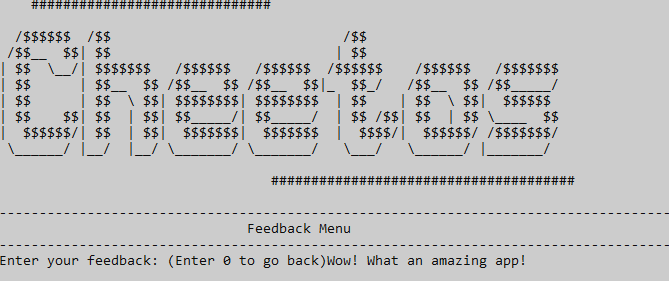
**Figure 23: Employee viewing ingredient request**

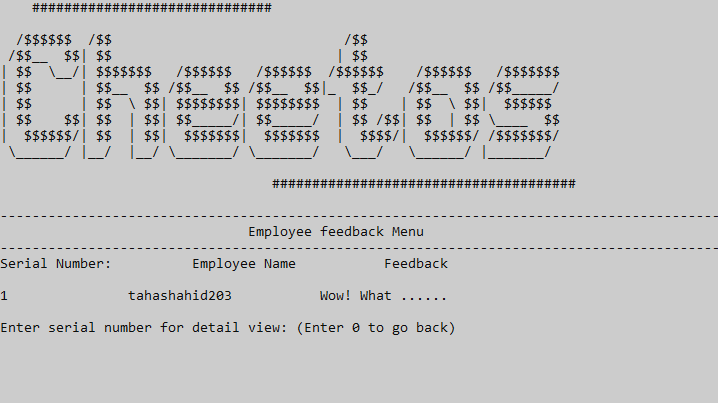
**Figure 24: Employee viewing his current orders**

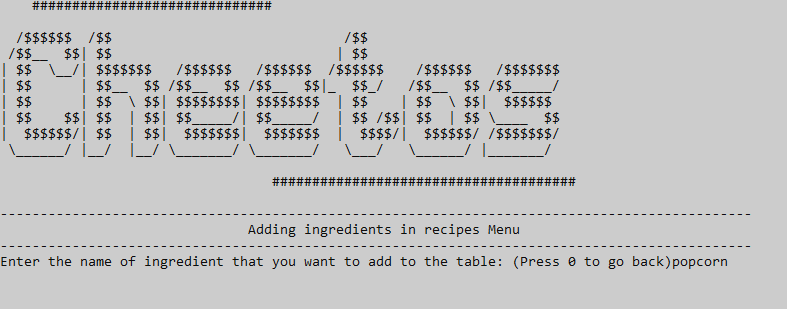


## **Figure 25: Employee viewing ingredients received**

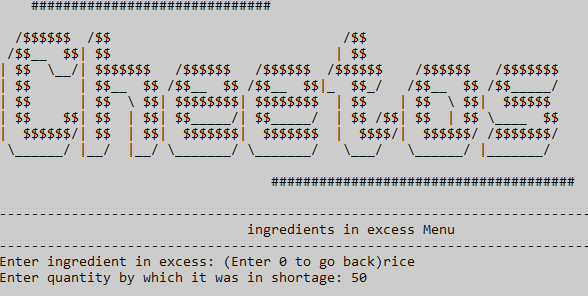
**Figure 26: Employee logging shortage of ingredients**

**Figure 27: Employee giving suggestions**

**Figure 28: Employee viewing all suggestions submitted**

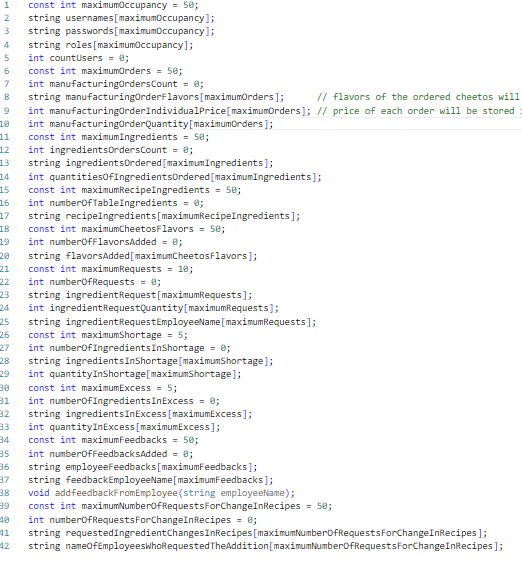


## **Figure 29: Employee requesting an addition in the recipe table**



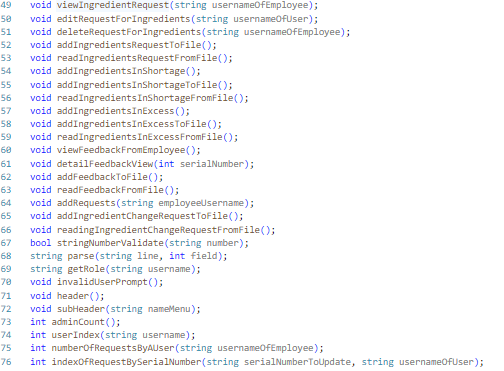
## **Figure 30: Employee logging extra ingredients**

# **Data Structures (Parallel Arrays)**

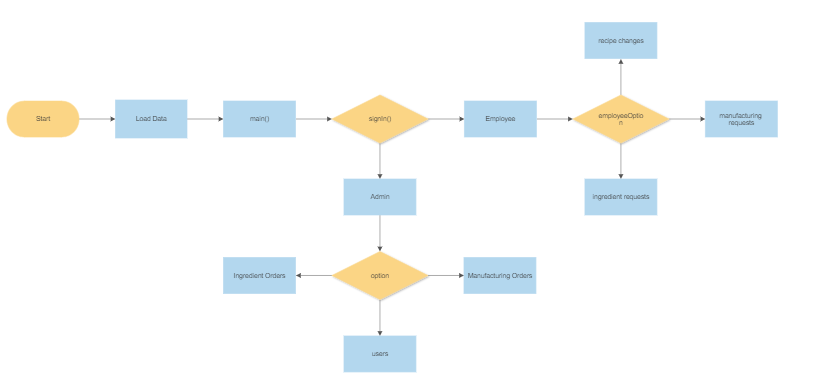
****

# **Function Prototypes**





# **Functions Working Flow**



# **Complete Code of the Business Application**

#include <iostream>

#include <fstream>

#include <windows.h>

#include <conio.h>

using namespace std;

/\* the arrays for usernames, passwords and roles must be global so that all functions can access them\*/

/\* The length of the array has been hardcored according to the capacity of the factory to house employees and admins \*/

const int maximumOccupancy = 50;

string usernames[maximumOccupancy];

string passwords[maximumOccupancy];

string roles[maximumOccupancy];

/\* Although the length of the username, password and roles arrays have been set according to the factory's capacity, we must only check

that portion of the array that is being used. Initially there's no user in the system. So, we must access

none of the elements of the arrays by setting countUsers = 0.

As the admin adds more users in the system, the countUsers variable will keep incrementing\*/

int countUsers = 0;

/\* To validate whether the user who has signed in actually exists in the system or not\*/

bool signInInterface(string username, string password);

/\*To show all the options that users get once signed in\*/

void adminInterface();

void employeeInterface();

/\*To see the choice that the admin has accessed\*/

string adminChoice();

string employeeChoice();

/\* function choosers based on the option that the user chose in the above functions \*/

bool adminFunctionChoosers(string option, string usernameOfUser);

bool employeeFunctionChoosers(string option, string usernameOfEmployee);

//................................................................................................................................................

//................................................................................................................................................

// ADMIN FUNCTIONS

//................................................................................................................................................

//................................................................................................................................................

/\* adding users \*/

// initial sign up

void initialSignUp();

/\* user adding menu \*/

bool addUserInterface(string usernameOfUser);

/\* To check whether a user already exists in the system \*/

bool userAuthenticate(string name);

/\* To add an authenticated user \*/

void addUser(string username, string password, string role);

/\*retrieving users \*/

void viewUsersInfo();

void viewSpecificUser(int index);

/\* Updating user info \*/

void updateUserInterface(string username);

void updateUserInfo(string username);

/\* deleting users \*/

bool deleteUserInterface(string username);

void deleteUser(string username);

// users in file

void addUsersToFile(); // to add users in file.

void loadUsersFromFile();

//-------------------------------------------------------------------------------------------------------------------------------------

// Manufacturing orders

// Adding orders

const int maximumOrders = 50;

int manufacturingOrdersCount = 0;

string manufacturingOrderFlavors[maximumOrders]; // flavors of the ordered cheetos will be stored here

int manufacturingOrderIndividualPrice[maximumOrders]; // price of each order will be stored in parallel here

int manufacturingOrderQuantity[maximumOrders]; // quantity of each order will be stored in parallel here

void addManufacturingOrder(); // function to add order to the arrays

bool orderPriceVerification(string price); // function to validate order price before converting them to int and storing in array

bool orderQuantityVerification(string quantity); // function to validate order quantity before converting them to int and storing in array

// view manufacturing orders

void viewManufacturingOrders();

// update manufacturing orders

void updateManufacturingOrders();

// delete manufacturing orders

void deleteManufacturingOrderInterface(); // interface

void deleteManufacturingOrders(int index); // deletor

// is a flavor in the system?

bool isFlavorInTheSystem(string flavor);

// storing mannufacturing orders in file

void addManufacturingOrdersToFile();

// reading manufacturing orders from file

void readManufacturingOrdersFromFile();

//..........................................................................................................................................................

// ingredients

const int maximumIngredients = 50;

int ingredientsOrdersCount = 0;

string ingredientsOrdered[maximumIngredients];

int quantitiesOfIngredientsOrdered[maximumIngredients];

// add ingredients

void addIngredients();

// view ingredients ordered

void viewIngredientsOrdered();

// update ingredients ordered

void updateIngredientsOrdered();

// delete ingredients ordered

void deleteIngredientsOrdered();

// validations

// is an ingredient in the recipe table?

bool isIngredientInTheTable(string ingredient);

// add ingredients to file

void addIngredientsOrdersToFile();

// read ingredients from file

void readIngredientsOrdersFromFile();

//-------------------------------------------------------------------------------------------------------------------------------------

// ingredients in table

const int maximumRecipeIngredients = 50;

int numberOfTableIngredients = 0;

string recipeIngredients[maximumRecipeIngredients];

// add ingredients to the table

void addIngredientsToTheTable();

// view ingredients in table

void viewIngredientsInTheTable();

// delete ingredients from the table

void deleteIngredientsFromTheTable();

// add table ingredients to file

void addIngredientsToFile();

// read ingredients from file

void readIngredientsFromFile();

//.......................................................................................................................................

// Cheetos Flavors

const int maximumCheetosFlavors = 50;

int numberOfFlavorsAdded = 0;

string flavorsAdded[maximumCheetosFlavors];

// adding cheetos flavors

void addCheetosFlavors();

// view flavors in system

void viewCheetosFlavors();

// deleting flavors

void deleteFlavorsFromSystem();

// add flavors to file

void addFlavorsToFile();

// read flavors from file

void readFlavorsFromFile();

// view recommended ingredients

void viewRecommendedIngredients();

// view excess ingredients

----void viewExcessIngredients();

//................................................................................................................................................

//................................................................................................................................................

// EMPLOYEE FUNCTIONS

//................................................................................................................................................

//................................................................................................................................................

const int maximumRequests = 10;

int numberOfRequests = 0;

string ingredientRequest[maximumRequests];

int ingredientRequestQuantity[maximumRequests];

string ingredientRequestEmployeeName[maximumRequests];

// requests.........................................................

// add request for ingredients

void uploadIngredientRequest(string usernameOfEmployee);

// view request for ingredients

void viewIngredientRequest(string usernameOfEmployee);

// edit requests for ingredients

void editRequestForIngredients(string usernameOfUser);

// delete requests for ingredients

void deleteRequestForIngredients(string usernameOfEmployee);

// add requests to file

void addIngredientsRequestToFile();

// read requests from file

void readIngredientsRequestFromFile();

// shortage of ingredients.....................................................

const int maximumShortage = 5;

int numberOfIngredientsInShortage = 0;

string ingredientsInShortage[maximumShortage];

int quantityInShortage[maximumShortage];

void addIngredientsInShortage();

// add shortage of ingredients to file

void addIngredientsInShortageToFile();

// read ingredients in shortage from file

void readIngredientsInShortageFromFile();

// excess of ingredients

const int maximumExcess = 5;

int numberOfIngredientsInExcess = 0;

string ingredientsInExcess[maximumExcess];

int quantityInExcess[maximumExcess];

void addIngredientsInExcess();

// add excess ingredients to file

void addIngredientsInExcessToFile();

// read excess ingredients from file

void readIngredientsInExcessFromFile();

// give feedback..............................................................

const int maximumFeedbacks = 50;

int numberOfFeedbacksAdded = 0;

string employeeFeedbacks[maximumFeedbacks];

string feedbackEmployeeName[maximumFeedbacks];

void addfeedbackFromEmployee(string employeeName);

// view feedback

void viewFeedbackFromEmployee();

void detailFeedbackView(int serialNumber);

// store feedback in file

void addFeedbackToFile();

// load feedback from file

void readFeedbackFromFile();

//..................................................................................

// request for change in recipe ingredients

const int maximumNumberOfRequestsForChangeInRecipes = 50;

int numberOfRequestsForChangeInRecipes = 0;

string requestedIngredientChangesInRecipes[maximumNumberOfRequestsForChangeInRecipes];

string nameOfEmployeesWhoRequestedTheAddition[maximumNumberOfRequestsForChangeInRecipes];

// adding requests

void addRequests(string employeeUsername);

// adding requests to file

void addIngredientChangeRequestToFile();

// reading requests from file

void readingIngredientChangeRequestFromFile();

//................................................................................................................................................

// helpful side functions

//................................................................................................................................................

bool stringNumberValidate(string number);

string parse(string line, int field);

/\* To get the role of a user as an admin or an employee once his credentials have been verified after signing in\*/

string getRole(string username);

/\* display invalid user prompt \*/

void invalidUserPrompt();

/\* The header of the system\*/

void header();

void subHeader(string nameMenu);

/\* admin option functions \*/

/\* a function to count the number of admins in the system. This number must always be greater than 0\*/

int adminCount();

/\* A function to return a user's index by using his username \*/

int userIndex(string username);

int numberOfRequestsByAUser(string usernameOfEmployee);

int indexOfRequestBySerialNumber(string serialNumberToUpdate, string usernameOfUser);

main()

{

readIngredientsFromFile();

loadUsersFromFile();

readIngredientsOrdersFromFile();

readManufacturingOrdersFromFile();

readIngredientsFromFile();

readFlavorsFromFile();

readIngredientsInShortageFromFile();

readIngredientsInExcessFromFile();

readFeedbackFromFile();

readingIngredientChangeRequestFromFile();

readIngredientsRequestFromFile();

if (countUsers == 0)

{

initialSignUp();

addUsersToFile();

}

/\* The original user of the system aka the first admin must be hardcored into the system\*/

while (true)

{

string username, password, role;

string exitAnswer = "N";

bool result = false; // result will check if the signed in credentials are valid or not

bool flagAdmin = false; // flagAdmin will allow us to quickly jump from choosing an admin option to the login page

bool flagEmployee = false; // similar to flagAdmin but for employees

system("cls");

header();

subHeader("Sign In");

cout << "Enter username: (Enter 0 to exit the program)";

cin >> username;

if (username == "0")

{

cout << "Are you sure you want to leave? (Y/N)";

cin >> exitAnswer;

if (exitAnswer == "Y" || exitAnswer == "y")

{

system("cls");

cout << "Thank you for your time" << endl;

break;

}

else

{

continue;

}

}

cout << "Enter password: ";

cin >> password;

/\* result is to check whether the signed in credentials are valid or not\*/

result = signInInterface(username, password);

if (result)

{

/\* if the user is valid, we obtain his role as an admin or employee and display the appropriate screens accordingly\*/

role = getRole(username);

if (role == "admin")

{

while (true)

{

system("cls");

header();

subHeader("Admin main");

adminInterface(); // show all available features for admin

string option = adminChoice(); // show admin's chosen option

/\* after getting the option chosen by the admin, the proper function will be called\*/

if (!adminFunctionChoosers(option, username))

{

flagAdmin = true;

break;

}

}

if (flagAdmin)

{

continue;

}

}

else if (role == "employee")

{

while (true)

{

system("cls");

header();

subHeader("Employee Main");

employeeInterface(); // show employee's available options

string employeeOption = employeeChoice(); // show employee's chosen option

/\* After getting the option chosen by the employee, the proper function will be displayed \*/

if (!employeeFunctionChoosers(employeeOption, username))

{

flagEmployee = true;

break;

}

}

if (flagEmployee)

{

continue;

}

}

/\*If the role is invalid, we tell user to try again\*/

else

{

invalidUserPrompt();

continue;

}

}

/\*If the user is invalid, we tell him/her to try again\*/

else

{

invalidUserPrompt();

continue;

}

}

}

bool signInInterface(string username, string password)

{

bool flag = false;

/\* As soon as the user is found in the system, we turn our flag true\*/

for (int i = 0; i < countUsers; i++)

{

if (username == usernames[i] && password == passwords[i])

{

flag = true;

}

}

return flag;

}

string getRole(string username)

{

/\*using the index of his username, we access his role in another parallel array\*/

for (int i = 0; i < countUsers; i++)

{

if (username == usernames[i])

{

return roles[i];

}

}

/\*if no role is found, we return invalid\*/

return "invalid";

}

void invalidUserPrompt()

{

char c;

cout << "Invalid user" << endl;

cout << endl;

cout << "Enter any key to try again" << endl;

getch();

}

void adminInterface()

{

/\*The menu that the admin will see upon signing in\*/

cout << "1 Create ingredients order: " << endl;

cout << "2 Create manufacturing order: " << endl;

cout << "3 View ingredients order: " << endl;

cout << "4 View manufacturing order: " << endl;

cout << "5 Update an ingredients order: " << endl;

cout << "6 Update a manufacturing order: " << endl;

cout << "7 Delete an ingredients order: " << endl;

cout << "8 Delete a manufacturing order: " << endl;

cout << "9 View recommended ingredients: " << endl;

cout << "10 View excess ingredients in factory stock:" << endl;

cout << "11 View ingredients table:" << endl;

cout << "12 Add ingredients to the table:" << endl;

cout << "13 View employees' suggestions" << endl;

cout << "14 Add user: " << endl;

cout << "15 Update user info" << endl;

cout << "16 View user info" << endl;

cout << "17 Delete user info" << endl;

cout << "18 Delete ingredient from the table" << endl;

cout << "19 Add a flavor into the system" << endl;

cout << "20 Delete a flavor from the system" << endl;

cout << "21 View flavors in system" << endl;

cout << endl;

cout << "Enter 0 to exit to login screen" << endl;

}

void employeeInterface()

{

/\*The menu that the emplyee will see upon signing in\*/

cout << "1 Upload a new request" << endl;

cout << "2 View requests: " << endl;

cout << "3 Edit requests " << endl;

cout << "4 Delete requests: " << endl;

cout << "5 View current orders: " << endl;

cout << "6 View ingredients received: " << endl;

cout << "7 Log shortage of ingredients " << endl;

cout << "8 Give a suggestion " << endl;

cout << "9 View your suggestions " << endl;

cout << "10 request section leader for editing recipes " << endl;

cout << "11 View ingredients recipe table " << endl;

cout << "12 View flavors added in the system " << endl;

cout << "13 Log excess of ingredients" << endl;

cout << endl;

cout << "Enter 0 to exit to login screen" << endl;

}

string adminChoice()

{

string choice;

char c;

while (true)

{

/\*the admin will enter his choice till he gets it right. Otherwise the system will keep asking for option\*/

cout << "Enter choice: ";

cin >> choice;

if (!(choice == "0" || choice == "1" || choice == "2" || choice == "3" || choice == "4" || choice == "5" || choice == "6" || choice == "7" || choice == "8" ||

choice == "9" || choice == "10" || choice == "11" || choice == "12" || choice == "13" || choice == "14" || choice == "15" || choice == "16" || choice == "17" || choice == "18" || choice == "19" || choice == "20" || choice == "21"))

{

cout << "That is not an available option." << endl;

cout << "Press any key to continue: " << endl;

getch();

continue;

}

return choice;

}

}

string employeeChoice()

{

string choice;

char c;

while (true)

{

/\*the employee will enter his choice till he gets it right. Otherwise the system will keep asking for option\*/

cout << "Enter choice: ";

cin >> choice;

if (!(choice == "0" || choice == "1" || choice == "2" || choice == "3" || choice == "4" || choice == "5" || choice == "6" || choice == "7" || choice == "8" || choice == "9" || choice == "10" || choice == "11" || choice == "12" || choice == "13"))

{

cout << "That is not an available option." << endl;

cout << "Press any key to continue: " << endl;

getch();

continue;

}

return choice;

}

}ٖ

bool adminFunctionChoosers(string option, string usernameOfUser) /\* the function chooser is bool because after choosing and excuting,

the control might return to the option \*/

{

char c;

if (option == "0")

{

return false;

}

else if (option == "1")

{

addIngredients();

}

else if (option == "2")

{

addManufacturingOrder();

}

else if (option == "3")

{

viewIngredientsOrdered();

}

else if (option == "4")

{

viewManufacturingOrders();

}

else if (option == "5")

{

updateIngredientsOrdered();

}

else if (option == "6")

{

updateManufacturingOrders();

}

else if (option == "7")

{

deleteIngredientsOrdered();

}

else if (option == "8")

{

deleteManufacturingOrderInterface();

}

else if (option == "9")

{

viewRecommendedIngredients();

}

else if (option == "10")

{

viewExcessIngredients();

}

else if (option == "11")

{

viewIngredientsInTheTable();

}

else if (option == "12")

{

addIngredientsToTheTable();

}

else if (option == "13")

{

viewFeedbackFromEmployee();

}

else if (option == "14")

{

if (addUserInterface(usernameOfUser))

{

cout << "User added successfully" << endl;

cout << "Press any key to continue" << endl;

getch();

}

else

{

system("cls");

header();

subHeader("User Adding");

cout << "Maximum occupancy reached" << endl;

cout << "Press any key to continue" << endl;

getch();

}

}

else if (option == "15")

{

updateUserInterface(usernameOfUser);

}

else if (option == "16")

{

viewUsersInfo();

}

else if (option == "17")

{

if (deleteUserInterface(usernameOfUser))

{

cout << "Press any key to continue" << endl;

getch();

}

else

{

return false;

}

}

else if (option == "18")

{

deleteIngredientsFromTheTable();

}

else if (option == "19")

{

addCheetosFlavors();

}

else if (option == "20")

{

deleteFlavorsFromSystem();

}

else if (option == "21")

{

viewCheetosFlavors();

}

return true;

}

bool employeeFunctionChoosers(string employeeOption, string usernameOfEmployee)

{

char c;

bool flag = true;

if (employeeOption == "0")

{

flag = false;

}

else if (employeeOption == "1")

{

uploadIngredientRequest(usernameOfEmployee);

}

else if (employeeOption == "2")

{

viewIngredientRequest(usernameOfEmployee);

}

else if (employeeOption == "3")

{

editRequestForIngredients(usernameOfEmployee);

}

else if (employeeOption == "4")

{

deleteRequestForIngredients(usernameOfEmployee);

}

else if (employeeOption == "5")

{

viewManufacturingOrders();

}

else if (employeeOption == "6")

{

viewIngredientsOrdered(); // ingredients ordered by the admin will be the ingredients recieved by the employee

}

else if (employeeOption == "7")

{

addIngredientsInShortage();

}

else if (employeeOption == "8")

{

addfeedbackFromEmployee(usernameOfEmployee);

}

else if (employeeOption == "9")

{

viewFeedbackFromEmployee();

}

else if (employeeOption == "10")

{

addRequests(usernameOfEmployee);

}

else if (employeeOption == "11")

{

viewIngredientsInTheTable();

}

else if (employeeOption == "12")

{

viewCheetosFlavors();

}

else if (employeeOption == "13")

{

addIngredientsInExcess();

}

return flag;

}

// Adding users to the system

// initial sign up

void initialSignUp()

{

string username, password;

system("cls");

cout << "Cheetos management system will be up and running shortly. Before we begin, however, let us set up a few things" << endl

<< endl;

cout << "This is the initial sign up of the system. After this sign up, the entered credentials will be automatically registered as admin." << endl

<< endl;

cout << "After that, the system has safeguards to ensure that atleast one admin stays in the system" << endl

<< endl

<< endl

<< endl;

cout << "Enter username: ";

cin >> username;

cout << "Enter password: ";

cin >> password;

addUser(username, password, "admin");

}

// later sign ups

bool addUserInterface(string usernameOfUser)

{

bool flag = true;

char c;

bool roleFlag = true;

bool usernameFlag = true;

bool passwordFlag = true;

string role, username, password;

while (true)

{

if (countUsers >= maximumOccupancy)

{

flag = false;

break;

}

if (flag)

{

system("cls");

header();

subHeader("User Adding");

if (roleFlag)

{

cout << "Enter type of user (employee, admin): ";

cin >> role;

}

if (usernameFlag)

{

cout << "Enter username: ";

cin >> username;

}

if (passwordFlag)

{

cout << "Enter password: ";

cin >> password;

}

if (!(role == "admin" || role == "employee"))

{

cout << "Invalid role" << endl;

cout << "Try again" << endl;

}

else

{

roleFlag = false;

}

if (username.length() < 5)

{

cout << "Username must have more than 5 letters." << endl;

cout << "Try again" << endl;

}

else if (!(userAuthenticate(username))) // to check whether the username is unique or not

{

cout << "Username already exists. Try again" << endl;

}

else

{

usernameFlag = false;

}

if (password.length() < 5)

{

cout << "password must have more than 5 letters." << endl;

cout << "Try again" << endl;

}

else

{

passwordFlag = false;

}

if (roleFlag || passwordFlag || usernameFlag)

{

cout << "Enter any key to continue" << endl;

getch();

}

else

{

break; // if all validations are passed, we break the loop and call the function to add the user in our array

}

}

}

if (flag)

{

addUser(username, password, role);

addUsersToFile();

}

return flag;

}

bool userAuthenticate(string username)

{

bool flag = true;

for (int i = 0; i < countUsers; i++)

{

if (usernames[i] == username)

{

flag = false;

break;

}

}

return flag;

}

void addUser(string username, string password, string role)

{

usernames[countUsers] = username;

passwords[countUsers] = password;

roles[countUsers] = role;

countUsers++; // increment the global variable that shows how many users have been added into the system.

}

// retrieving users from the system

void viewUsersInfo()

{

string option;

char c;

while (true)

{

bool flag = false;

system("cls");

header();

subHeader("Users Info");

cout << "Sr. no"

<< "\t"

<< "\t"

<< "username" << endl;

for (int i = 0; i < countUsers; i++)

{

cout << i + 1 << "\t"

<< "\t" << usernames[i] << endl;

}

cout << "Select user to view all details: (Enter 0 to go back)";

cin >> option;

for (int i = 0; i < option.length(); i++)

{

if (!(option[i] >= 48 && option[i] <= 57))

{

cout << "Option must only contain numbers" << endl;

cout << "Enter any key to continue" << endl;

getch();

flag = true;

break;

}

}

if (flag)

{

continue;

}

int option2 = stoi(option);

if (option2 > 0 && option2 <= countUsers)

{

option2--;

viewSpecificUser(option2);

}

else if (option2 == 0)

{

break;

}

else

{

cout << "You entered the wrong option. Try again" << endl;

getch();

continue;

}

}

}

void viewSpecificUser(int option)

{

char c;

system("cls");

header();

subHeader("User Info");

cout << "Username: " << usernames[option] << endl;

if (roles[option] != "admin")

{

cout << "Password: " << passwords[option] << endl;

}

cout << "Role: " << roles[option] << endl;

cout << "Press any key to go back" << endl;

getch();

}

// update user info

void updateUserInterface(string username)

{

char c;

string usernameToUpdate, roleToUpdate;

while (true)

{

system("cls");

header();

subHeader("Update User");

cout << "Enter the username you want to update: (Enter 0 to go back)";

cin >> usernameToUpdate;

if (usernameToUpdate == "0")

{

break;

}

for (int i = 0; i < countUsers; i++)

{

if (usernameToUpdate == usernames[i])

{

roleToUpdate = roles[i];

break;

}

}

if (roleToUpdate == "employee" || usernameToUpdate == username)

{

updateUserInfo(usernameToUpdate);

}

else if (roleToUpdate == "admin")

{

cout << "You cannot update an admin account without signing into it" << endl;

cout << "press any key to continue" << endl;

getch();

continue;

}

else

{

cout << "Username does not exist" << endl;

cout << "press any key to continue" << endl;

getch();

continue;

}

}

}

void updateUserInfo(string username)

{

char c;

string newUsername, newPassword, oldPassword, newRole;

while (true)

{

system("cls");

header();

subHeader("Update User");

cout << "Enter new username: ";

cin >> newUsername;

cout << "Enter new password: ";

cin >> newPassword;

cout << "Enter new role: ";

cin >> newRole;

if (newUsername != username)

{

if (!(userAuthenticate(newUsername)))

{

cout << "Username already exists. Try again" << endl;

getch();

continue;

}

}

if (newRole == "employee")

{

if (adminCount() == 1 && getRole(username) == "admin")

{

cout << "You cannot change your role to employee because there will be no more admins left" << endl;

cout << "press any key to continue" << endl;

getch();

newRole = "admin";

}

}

for (int i = 0; i < countUsers; i++)

{

if (username == usernames[i])

{

usernames[i] = newUsername;

passwords[i] = newPassword;

roles[i] = newRole;

addUsersToFile();

break;

}

}

break;

}

}

// delete user

bool deleteUserInterface(string username)

{

string usernameToBeDeleted;

char c;

bool flag = true;

while (true)

{

system("cls");

header();

subHeader("User Deleting");

cout << "Enter username: (Press 0 to go back)";

cin >> usernameToBeDeleted;

if (usernameToBeDeleted == "0")

{

break;

}

if (!(userAuthenticate(usernameToBeDeleted)))

{

if (roles[userIndex(usernameToBeDeleted)] == "admin")

{

if (usernameToBeDeleted != username)

{

cout << "You cannot delete an admin without signin into his account" << endl;

cout << "Press any key to try again" << endl;

getch();

continue;

}

else

{

if (adminCount() == 1)

{

cout << "You cannot delete this account or there will be no more users left" << endl;

cout << "Press any key to continue" << endl;

getch();

continue;

}

else

{

flag = false;

}

}

}

deleteUser(usernameToBeDeleted);

cout << "Press any key to continue" << endl;

getch();

}

else

{

cout << "Username does not exist. Please try again" << endl;

cout << "Press any key to continue" << endl;

getch();

continue;

}

}

return flag;

}

void deleteUser(string username)

{

int index = userIndex(username);

for (int i = index; i < countUsers - 1; i++)

{

usernames[i] = usernames[i + 1];

passwords[i] = passwords[i + 1];

roles[i] = roles[i + 1];

}

usernames[countUsers - 1] = "temporary123";

passwords[countUsers - 1] = "temporary123";

roles[countUsers - 1] = "temporary123";

countUsers--;

addUsersToFile();

}

int adminCount()

{

int count = 0;

for (int i = 0; i < countUsers; i++)

{

if (roles[i] == "admin")

{

count++;

}

}

return count;

}

int userIndex(string username)

{

for (int i = 0; i < countUsers; i++)

{

if (usernames[i] == username)

{

return i;

}

}

return -1;

}

// adding users to the file

void addUsersToFile()

{

fstream fileUser;

fileUser.open("users.txt", ios::out);

for (int i = 0; i < countUsers; i++)

{

fileUser << usernames[i] << "," << passwords[i] << "," << roles[i] << endl;

}

fileUser.close();

}

// loading users from sile

void loadUsersFromFile()

{

string line;

fstream fileUser;

fileUser.open("users.txt", ios::in);

while (!fileUser.eof())

{

getline(fileUser, line);

if (line == "")

{

continue;

}

usernames[countUsers] = parse(line, 1);

passwords[countUsers] = parse(line, 2);

roles[countUsers] = parse(line, 3);

countUsers++;

}

fileUser.close();

}

//--------------------------------------------------------------------------------------------------------------------------------------------------------

// manufacturing orders

// adding

void addManufacturingOrder()

{

string choice, numberOfOrders, temporaryQuantity, temporaryPrice, temporaryFlavor;

while (true)

{

system("cls");

header();

subHeader("Manufacturing orders");

cout << "Enter the number of orders that you wish to enter: (Enter 0 to go back)" << endl;

cout << "You can enter " << maximumOrders - manufacturingOrdersCount << " orders";

cin >> numberOfOrders;

if (numberOfOrders == "0")

{

break;

}

if (!stringNumberValidate(numberOfOrders))

{

cout << "Enter a correct number!" << endl;

cout << "Press any key to continue" << endl;

getch();

continue;

}

if (stoi(numberOfOrders) + manufacturingOrdersCount > 50)

{

cout << "You cannot add so many orders. " << endl;

cout << "Enter under " << maximumOrders - manufacturingOrdersCount << " orders" << endl;

cout << "Press any key to continue" << endl;

getch();

continue;

}

for (int i = 0; i < stoi(numberOfOrders); i++)

{

cout << "Enter Flavor: (Press 0 to exit)";

cin >> temporaryFlavor;

if (temporaryFlavor == "0")

{

break;

}

if (isFlavorInTheSystem(temporaryFlavor))

{

manufacturingOrderFlavors[i] = temporaryFlavor;

}

else

{

system("cls");

cout << "This flavor has not been stored in the system. Press any key to try again" << endl;

getch();

i--;

continue;

}

cout << "Enter quantity: ";

cin >> temporaryQuantity;

if (orderQuantityVerification(temporaryQuantity) && stringNumberValidate(temporaryQuantity))

{

manufacturingOrderQuantity[manufacturingOrdersCount] = stoi(temporaryQuantity);

}

else

{

cout << "Not a valid quantity. Press any key to try again" << endl;

getch();

i--;

continue;

}

cout << "Enter price: ";

cin >> temporaryPrice;

if (orderPriceVerification(temporaryPrice) && stringNumberValidate(temporaryPrice))

{

manufacturingOrderIndividualPrice[manufacturingOrdersCount] = stoi(temporaryPrice);

}

else

{

cout << "Not a valid price. Press any key to try again" << endl;

getch();

i--;

continue;

}

manufacturingOrdersCount++;

}

addManufacturingOrdersToFile();

}

}

bool orderPriceVerification(string price)

{

return true;

}

bool orderQuantityVerification(string quantity)

{

return true;

}

// view manufacturing orders

void viewManufacturingOrders()

{

system("cls");

header();

subHeader("Manufacturing Order view");

cout << "Serial number:"

<< "\t"

<< "Flavor:"

<< "\t\t"

<< "Quantity"

<< "\t\t"

<< "Price" << endl

<< endl;

for (int i = 0; i < manufacturingOrdersCount; i++)

{

cout << i + 1 << "\t\t\t" << manufacturingOrderFlavors[i] << "\t\t\t" << manufacturingOrderQuantity[i] << "\t\t\t" << manufacturingOrderIndividualPrice[i] << endl;

}

cout << endl;

cout << "Press any key to continue:" << endl;

getch();

}

// update manufaturing orders

void updateManufacturingOrders()

{

string serialNumberToUpdate, temporaryQuantity, temporaryPrice, temporaryFlavor;

bool flavorFlag = true;

bool quantityFlag = true;

bool priceFlag = true;

bool serialNumberFlag = true;

while (true)

{

system("cls");

header();

subHeader("Update Order");

if (serialNumberFlag)

{

cout << "Enter serial number of the order you want to update: ";

cout << "Enter 0 to go back";

cin >> serialNumberToUpdate;

}

if (serialNumberToUpdate == "0")

{

break;

}

if (stringNumberValidate(serialNumberToUpdate))

{

if (stoi(serialNumberToUpdate) > 0 && stoi(serialNumberToUpdate) <= manufacturingOrdersCount)

{

serialNumberFlag = false;

if (flavorFlag)

{

cout << "Enter new flavor: ";

cin >> temporaryFlavor;

if (isFlavorInTheSystem(temporaryFlavor))

{

manufacturingOrderFlavors[stoi(serialNumberToUpdate) - 1] = temporaryFlavor;

}

}

if (quantityFlag)

{

cout << "Enter new quantity: ";

cin >> temporaryQuantity;

}

if (priceFlag)

{

cout << "Enter new price: ";

cin >> temporaryPrice;

}

if (stringNumberValidate(temporaryQuantity) && orderQuantityVerification(temporaryQuantity))

{

manufacturingOrderQuantity[stoi(serialNumberToUpdate) - 1] = stoi(temporaryQuantity);

quantityFlag = false;

}

if (stringNumberValidate(temporaryPrice) && orderPriceVerification(temporaryPrice))

{

manufacturingOrderIndividualPrice[stoi(serialNumberToUpdate) - 1] = stoi(temporaryPrice);

priceFlag = false;

}

flavorFlag = false; // future validations for flavor will be called here

if (serialNumberFlag || flavorFlag || priceFlag || quantityFlag)

{

system("cls");

cout << "Enter correct information. Press any key to continue" << endl;

getch();

continue;

}

else

{

addManufacturingOrdersToFile();

break;

}

}

else

{

system("cls");

cout << "Enter correct information. Press any key to continue" << endl;

getch();

continue;

}

}

else

{

system("cls");

cout << "Enter correct information. Press any key to continue" << endl;

getch();

continue;

}

}

}

// delete manufacturing orders

// interface with validations

void deleteManufacturingOrderInterface()

{

string serialNumberToDelete;

while (true)

{

system("cls");

header();

subHeader("Delete Manufacturing orders");

cout << "Enter serial number of order you want to delete. (Press 0 to exit)" << endl;

cin >> serialNumberToDelete;

if (serialNumberToDelete == "0")

{

break;

}

if (stringNumberValidate(serialNumberToDelete))

{

if (stoi(serialNumberToDelete) > 0 && stoi(serialNumberToDelete) <= manufacturingOrdersCount)

{

deleteManufacturingOrders(stoi(serialNumberToDelete) - 1);

}

else

{

cout << "Enter valid number. Press any key to continue" << endl;

getch();

}

}

else

{

cout << "Enter valid number. Press any key to continue" << endl;

getch();

}

}

}

// deletor

void deleteManufacturingOrders(int index)

{

for (int i = index; i < manufacturingOrdersCount - 1; i++)

{

manufacturingOrderFlavors[i] = manufacturingOrderFlavors[i + 1];

manufacturingOrderIndividualPrice[i] = manufacturingOrderIndividualPrice[i + 1];

manufacturingOrderQuantity[i] = manufacturingOrderQuantity[i + 1];

}

manufacturingOrdersCount--;

addManufacturingOrdersToFile();

}

// validations

// is a manufacturing flavor in the system?

bool isFlavorInTheSystem(string flavor)

{

bool flag = false;

for (int i = 0; i < numberOfFlavorsAdded; i++)

{

if (flavor == flavorsAdded[i])

{

flag = true;

break;

}

}

return flag;

}

// adding manufacturing orders to file

void addManufacturingOrdersToFile()

{

fstream file;

file.open("manufacturingOrders.txt", ios::out);

for (int i = 0; i < manufacturingOrdersCount; i++)

{

file << manufacturingOrderFlavors[i] << "," << manufacturingOrderQuantity[i] << "," << manufacturingOrderIndividualPrice[i] << endl;

}

file.close();

}

// reading manufacturing orders from file

void readManufacturingOrdersFromFile()

{

string line;

fstream file;

file.open("manufacturingOrders.txt", ios::in);

while (!file.eof())

{

getline(file, line);

if (line == "")

{

continue;

}

manufacturingOrderFlavors[manufacturingOrdersCount] = parse(line, 1);

manufacturingOrderQuantity[manufacturingOrdersCount] = stoi(parse(line, 2));

manufacturingOrderIndividualPrice[manufacturingOrdersCount] = stoi(parse(line, 3));

manufacturingOrdersCount++;

}

file.close();

}

//.......................................................................................................................................

// ingredients

// add ingredients

void addIngredients()

{

string ingredientTobeOrdered, quantityOfIngredient;

while (true)

{

system("cls");

header();

subHeader("Adding ingredients");

cout << "Enter the name of the ingredient that you want to order: (Press 0 to go back)";

cin >> ingredientTobeOrdered;

if (ingredientTobeOrdered == "0")

{

break;

}

// ingredients' validatons will be called here in the future

if (!isIngredientInTheTable(ingredientTobeOrdered))

{

system("cls");

cout << "This ingredient is not in the recipe table" << endl;

cout << "Press any key to try again" << endl;

getch();

continue;

}

ingredientsOrdered[ingredientsOrdersCount] = ingredientTobeOrdered;

cout << "Enter quantity of the ingredient to be ordered: ";

cin >> quantityOfIngredient;

if (stringNumberValidate(quantityOfIngredient))

{

quantitiesOfIngredientsOrdered[ingredientsOrdersCount] = stoi(quantityOfIngredient);

}

else

{

cout << "Enter valid quantity. Press any key to try again" << endl;

getch();

continue;

}

addIngredientsOrdersToFile();

ingredientsOrdersCount++;

}

}

// view ingredients ordered

void viewIngredientsOrdered()

{

system("cls");

header();

subHeader("View ingredients");

cout << "Serial number"

<< "\t\t"

<< "Ingredients"

<< "\t\t"

<< "Qunatity ordered" << endl

<< endl;

for (int i = 0; i < ingredientsOrdersCount; i++)

{

cout << i + 1 << "\t\t" << ingredientsOrdered[i] << "\t\t" << quantitiesOfIngredientsOrdered[i] << endl;

}

cout << "Press any key to continue: ";

getch();

}

// update ingredients ordered

void updateIngredientsOrdered()

{

string serialNumberToUpdate, newTemporaryIngredeint, newTemporaryQuantity;

bool serialNumberFlag = true;

bool newIngredientFlag = true;

bool newIngredientQuantityFlag = true;

while (true)

{

system("cls");

header();

subHeader("Update Ingredients");

if (!newIngredientFlag && !newIngredientQuantityFlag)

{

serialNumberFlag = true;

addManufacturingOrdersToFile();

}

if (serialNumberFlag)

{

cout << "Enter serial number of the ingredient you want to update: (Enter 0 to go back)";

cin >> serialNumberToUpdate;

}

if (serialNumberToUpdate == "0")

{

break;

}

if (stringNumberValidate(serialNumberToUpdate))

{

if (stoi(serialNumberToUpdate) > 0 && stoi(serialNumberToUpdate) < ingredientsOrdersCount)

{

serialNumberFlag = false;

if (newIngredientFlag)

{

cout << "Enter new ingredient: ";

cin >> newTemporaryIngredeint;

}

// ingredients validation will be added here

if (!isIngredientInTheTable(newTemporaryIngredeint))

{

system("cls");

cout << "This ingredient is not in the recipe table" << endl;

cout << "Press any key to try again" << endl;

getch();

continue;

}

newIngredientFlag = false;

ingredientsOrdered[stoi(serialNumberToUpdate) - 1] = newTemporaryIngredeint;

cout << "Enter new quantity: ";

cin >> newTemporaryQuantity;

if (stringNumberValidate(newTemporaryQuantity))

{

quantitiesOfIngredientsOrdered[stoi(serialNumberToUpdate) - 1] = stoi(newTemporaryQuantity);

newIngredientQuantityFlag = false;

}

}

else

{

cout << "Enter valid number. Press any key to try again";

getch();

}

}

else

{

cout << "Enter valid number. Press any key to try again";

getch();

}

}

}

// delete ingredients

void deleteIngredientsOrdered()

{

string serialNumberToDelete;

while (true)

{

system("cls");

header();

subHeader("Delete ingredinets");

cout << "Enter serial number of the ingredient you want to delete: (Enter 0 to go back)";

cin >> serialNumberToDelete;

if (serialNumberToDelete == "0")

{

break;

}

if (stringNumberValidate(serialNumberToDelete))

{

if (stoi(serialNumberToDelete) > 0 && stoi(serialNumberToDelete) <= ingredientsOrdersCount)

{

for (int i = stoi(serialNumberToDelete); i < ingredientsOrdersCount - 1; i++)

{

ingredientsOrdered[i] = ingredientsOrdered[i + 1];

quantitiesOfIngredientsOrdered[i] = quantitiesOfIngredientsOrdered[i + 1];

}

ingredientsOrdersCount--;

addManufacturingOrdersToFile();

}

else

{

cout << "Invalid number. Press any key to try again";

continue;

}

}

else

{

cout << "Invalid number. Press any key to try again";

continue;

}

}

}

// validations

// is an ingredient in the recipe table?

bool isIngredientInTheTable(string ingredient)

{

bool flag = false;

for (int i = 0; i < numberOfTableIngredients; i++)

{

if (ingredient == recipeIngredients[i])

{

flag = true;

break;

}

}

return flag;

}

// add ingredients to file

void addIngredientsOrdersToFile()

{

fstream file;

file.open("ingredientsOrders.txt", ios::out);

for (int i = 0; i < ingredientsOrdersCount; i++)

{

file << ingredientsOrdered[i] << "," << quantitiesOfIngredientsOrdered[i] << endl;

}

file.close();

}

// read ingredients from file

void readIngredientsOrdersFromFile()

{

fstream file;

string line;

file.open("ingredientsOrders.txt", ios::in);

while (!file.eof())

{

getline(file, line);

if (line == "")

{

continue;

}

ingredientsOrdered[ingredientsOrdersCount] = parse(line, 1);

quantitiesOfIngredientsOrdered[ingredientsOrdersCount] = stoi(parse(line, 2));

ingredientsOrdersCount++;

}

file.close();

}

//--------------------------------------------------------------------------------------------------

// Add ingredients to the table

void addIngredientsToTheTable()

{

string tableIngredient;

while (true)

{

system("cls");

header();

subHeader("Ingredients Table");

cout << "Enter name of ingredient that you want to add: " << endl;

cout << "(Press 0 to go back)" << endl;

cin >> tableIngredient;

if (tableIngredient == "0")

{

break;

}

recipeIngredients[numberOfTableIngredients] = tableIngredient;

numberOfTableIngredients++;

addIngredientsToFile();

}

}

// view ingredients in the table

void viewIngredientsInTheTable()

{

system("cls");

header();

subHeader("Table Ingredients");

cout << "Serial Number"

<< "\t\t"

<< "Ingredient" << endl;

for (int i = 0; i < numberOfTableIngredients; i++)

{

cout << i + 1 << "\t\t" << recipeIngredients[i] << endl;

}

cout << "Press any key to continue" << endl;

getch();

}

// delete ingredients from the table

void deleteIngredientsFromTheTable()

{

string serialNumberToDelete;

while (true)

{

system("cls");

header();

subHeader("Deleting Table Ingredients");

cout << "Enter serial number (from the recipee table) of the ingredient you want to delete: " << endl;

cout << "Enter 0 to go back";

cin >> serialNumberToDelete;

if (serialNumberToDelete == "0")

{

break;

}

if (stringNumberValidate(serialNumberToDelete))

{

int index = stoi(serialNumberToDelete) - 1;

if (index >= 0 && index < numberOfTableIngredients)

{

for (int i = index; i < numberOfTableIngredients - 1; i++)

{

recipeIngredients[i] = recipeIngredients[i + 1];

}

numberOfTableIngredients--;

addIngredientsToFile();

}

else

{

cout << "Enter a valid number. Press any key to try again" << endl;

getch();

}

}

else

{

cout << "Enter a valid number. Press any key to try again" << endl;

getch();

}

}

}

// add table ingredients to file

void addIngredientsToFile()

{

fstream file;

file.open("tableIngredients.txt", ios::out);

for (int i = 0; i < numberOfTableIngredients; i++)

{

file << recipeIngredients[i] << endl;

}

file.close();

}

// read ingredients from file

void readIngredientsFromFile()

{

fstream file;

string line;

file.open("tableIngredients.txt", ios::in);

while (!file.eof())

{

getline(file, line);

if (line == "")

{

continue;

}

recipeIngredients[numberOfTableIngredients] = line;

numberOfTableIngredients++;

}

file.close();

}

//................................................................................................................................................

// cheetos flavors

// adding

void addCheetosFlavors()

{

string cheetosFlavor;

while (true)

{

system("cls");

header();

subHeader("Cheetos Flavor");

cout << "Enter name of flavor that you want to add: " << endl;

cout << "(Press 0 to go back)" << endl;

cin >> cheetosFlavor;

if (cheetosFlavor == "0")

{

break;

}

flavorsAdded[numberOfFlavorsAdded] = cheetosFlavor;

numberOfFlavorsAdded++;

addFlavorsToFile();

}

}

// viewing

void viewCheetosFlavors()

{

system("cls");

header();

subHeader("Cheetos Flavors");

cout << "Serial Number"

<< "\t\t"

<< "Flavor" << endl;

for (int i = 0; i < numberOfFlavorsAdded; i++)

{

cout << i + 1 << "\t\t" << flavorsAdded[i] << endl;

}

cout << "Press any key to continue" << endl;

getch();

}

// delete flavors

void deleteFlavorsFromSystem()

{

string serialNumberToDelete;

while (true)

{

system("cls");

header();

subHeader("Deleting Cheetos Flavors");

cout << "Enter serial number (from the flavor table) of the flavor you want to delete: " << endl;

cout << "Enter 0 to go back";

cin >> serialNumberToDelete;

if (serialNumberToDelete == "0")

{

break;

}

if (stringNumberValidate(serialNumberToDelete))

{

int index = stoi(serialNumberToDelete) - 1;

if (index >= 0 && index < numberOfFlavorsAdded)

{

for (int i = index; i < numberOfFlavorsAdded - 1; i++)

{

flavorsAdded[i] = flavorsAdded[i + 1];

}

numberOfFlavorsAdded--;

addFlavorsToFile();

}

else

{

cout << "Enter a valid number. Press any key to try again" << endl;

getch();

}

}

else

{

cout << "Enter a valid number. Press any key to try again" << endl;

getch();

}

}

}

// add flavors to file

void addFlavorsToFile()

{

fstream file;

file.open("systemFlavors.txt", ios::out);

for (int i = 0; i < numberOfFlavorsAdded; i++)

{

file << flavorsAdded[i] << endl;

}

file.close();

}

// read flavors from file

void readFlavorsFromFile()

{

fstream file;

string line;

file.open("systemFlavors.txt", ios::in);

while (!file.eof())

{

getline(file, line);

if (line == "")

{

continue;

}

flavorsAdded[numberOfFlavorsAdded] = line;

numberOfFlavorsAdded++;

}

file.close();

}

// viewing recommended products

void viewRecommendedIngredients()

{

system("cls");

header();

subHeader("Recommended Products");

for (int i = 0; i < numberOfRequests; i++)

{

cout << i + 1 << "\t\t" << ingredientRequest[i] << "\t\t" << ingredientRequestQuantity[i] << endl;

}

for (int i = 0; i < numberOfIngredientsInShortage; i++)

{

cout << numberOfRequests + i << "\t\t" << ingredientsInShortage[i] << "\t\t" << quantityInShortage[i] << endl;

}

cout << "Press any key to continue" << endl;

getch();

}

// view excess ingredients

void viewExcessIngredients()

{

system("cls");

header();

subHeader("Excess Products");

for (int i = 0; i < numberOfIngredientsInExcess; i++)

{

cout << i + 1 << "\t\t" << ingredientsInExcess[i] << "\t\t" << quantityInExcess[i] << endl;

}

cout << "Press any key to continue" << endl;

getch();

}

//................................................................................................................................................

//................................................................................................................................................

// EMPLOYEE FUNCTIONS

//................................................................................................................................................

//................................................................................................................................................

// adding ingredient request

void uploadIngredientRequest(string usernameOfEmployee)

{

string requestedIngredient, requestedQuantity;

while (true)

{

system("cls");

header();

subHeader("Ingredient Request");

cout << "Enter requested ingredient: (Press 0 to go back)";

cin >> requestedIngredient;

if (requestedIngredient == "0")

{

break;

}

if (isIngredientInTheTable(requestedIngredient))

{

ingredientRequest[numberOfRequests] = requestedIngredient;

}

else

{

system("cls");

cout << "This ingredient is not in the recipe table. Press any key to try again" << endl;

getch();

continue;

}

cout << "Enter requested ingredient's quantity: ";

cin >> requestedQuantity;

if (stringNumberValidate(requestedQuantity))

{

ingredientRequestQuantity[numberOfRequests] = stoi(requestedQuantity);

}

else

{

system("cls");

cout << "Enter valid quantity. Press any key to try again" << endl;

getch();

continue;

}

ingredientRequestEmployeeName[numberOfRequests] = usernameOfEmployee;

numberOfRequests++;

addIngredientsRequestToFile();

}

}

// viewing ingredient request

void viewIngredientRequest(string usernameOfEmployee)

{

system("cls");

header();

subHeader("Employee requests");

int count = 1;

cout << "Serial Number"

<< "\t\t"

<< "Ingredient"

<< "\t\t"

<< "Quantity" << endl

<< endl;

for (int i = 0; i < numberOfRequests; i++)

{

if (ingredientRequestEmployeeName[i] == usernameOfEmployee)

{

cout << count << "\t\t" << ingredientRequest[i] << "\t\t" << ingredientRequestQuantity[i] << endl;

count++;

}

}

cout << endl;

cout << "Press any key to continue" << endl;

getch();

}

// edit request for ingredient

void editRequestForIngredients(string usernameOfUser)

{

string serialNumberToUpdate, newIngredientToOrder, newIngredientQuantity;

int indexOfRequest;

while (true)

{

system("cls");

header();

subHeader("Editing request");

cout << "Enter serial number of the request you want to edit: (Press 0 to go back)" << endl;

cin >> serialNumberToUpdate;

if (serialNumberToUpdate == "0")

{

break;

}

if (stringNumberValidate(serialNumberToUpdate))

{

if (stoi(serialNumberToUpdate) > 0 && stoi(serialNumberToUpdate) <= numberOfRequestsByAUser(usernameOfUser))

{

indexOfRequest = indexOfRequestBySerialNumber(serialNumberToUpdate, usernameOfUser);

cout << "Enter new ingredient to order: ";

cin >> newIngredientToOrder;

if (isIngredientInTheTable(newIngredientToOrder))

{

ingredientRequest[indexOfRequest] = newIngredientToOrder;

}

else

{

system("cls");

cout << "This ingredient is not in the recipe table. Press any key to try again" << endl;

getch();

continue;

}

cout << "Enter new quantity: ";

cin >> newIngredientQuantity;

if (stringNumberValidate(newIngredientQuantity))

{

ingredientRequestQuantity[indexOfRequest] = stoi(newIngredientQuantity);

addIngredientsRequestToFile();

}

else

{

system("cls");

cout << "Enter a valid number. Press any key to try again";

getch();

continue;

}

}

else

{

system("cls");

cout << "Enter a valid number. Press any key to try again.";

getch();

continue;

}

}

else

{

system("cls");

cout << "Enter a valid number. Press any key to try again";

getch();

continue;

}

}

}

// delete ingredient request

void deleteRequestForIngredients(string usernameOfEmployee)

{

string serialNumberToDelete;

int indexOfRequest;

while (true)

{

system("cls");

header();

subHeader("Ingredient request deletion");

cout << "Enter serial number of order that you want to delete: (Press 0 to go back)";

cin >> serialNumberToDelete;

if (serialNumberToDelete == "0")

{

break;

}

if (stringNumberValidate(serialNumberToDelete))

{

if (stoi(serialNumberToDelete) > 0 && stoi(serialNumberToDelete) <= numberOfRequestsByAUser(usernameOfEmployee))

{

indexOfRequest = indexOfRequestBySerialNumber(serialNumberToDelete, usernameOfEmployee);

for (int i = indexOfRequest; i < numberOfRequests; i++)

{

ingredientRequest[i] = ingredientRequest[i + 1];

ingredientRequestQuantity[i] = ingredientRequestQuantity[i + 1];

ingredientRequestEmployeeName[i] = ingredientRequestEmployeeName[i + 1];

}

numberOfRequests--;

addIngredientsRequestToFile();

}

else

{

system("cls");

cout << "Enter valid number. Press any key to try again." << endl;

getch();

continue;

}

}

else

{

system("cls");

cout << "Enter valid number. Press any key to try again." << endl;

getch();

continue;

}

}

}

// add ingredients requests to file

void addIngredientsRequestToFile()

{

fstream file;

file.open("employeeIngredientsRequest.txt", ios::out);

for (int i = 0; i < numberOfRequests; i++)

{

file << ingredientRequest[i] << "," << ingredientRequestQuantity[i] << "," << ingredientRequestEmployeeName[i] << endl;

}

file.close();

}

void readIngredientsRequestFromFile()

{

fstream file;

string line;

file.open("employeeIngredientsRequest.txt", ios::in);

while (!file.eof())

{

getline(file, line);

if (line == "")

{

continue;

}

ingredientRequest[numberOfRequests] = parse(line, 1);

ingredientRequestQuantity[numberOfRequests] = stoi(parse(line, 2));

ingredientRequestEmployeeName[numberOfRequests] = parse(line, 3);

numberOfRequests++;

}

file.close();

}

// add shortage of ingredients.....................................................................

void addIngredientsInShortage()

{

string ingredientInShortage, quantityByWhichItWasShort;

while (true)

{

system("cls");

header();

subHeader("ingredients in shortage");

cout << "Enter ingredient in shortage: (Enter 0 to go back)";

cin >> ingredientInShortage;

if (ingredientInShortage == "0")

{

break;

}

if (isIngredientInTheTable(ingredientInShortage))

{

ingredientsInShortage[numberOfIngredientsInShortage] = ingredientInShortage;

}

else

{

system("cls");

cout << "This ingredient is not in the recipe table. Press any key to try again." << endl;

getch();

continue;

}

cout << "Enter quantity by which it was in shortage: ";

cin >> quantityByWhichItWasShort;

if (stringNumberValidate(quantityByWhichItWasShort))

{

quantityInShortage[numberOfIngredientsInShortage] = stoi(quantityByWhichItWasShort);

numberOfIngredientsInShortage++;

addIngredientsInShortageToFile();

}

else

{

system("cls");

cout << "Enter valid quantity. Press any key to try again." << endl;

getch();

continue;

}

}

}

// add ingredients in shortage to file

void addIngredientsInShortageToFile()

{

fstream file;

file.open("shortageOfIngredients.txt", ios::out);

for (int i = 0; i < numberOfIngredientsInShortage; i++)

{

file << ingredientsInShortage[i] << "," << quantityInShortage[i] << endl;

}

file.close();

}

// read ingredients in shortage from file

void readIngredientsInShortageFromFile()

{

fstream file;

string line;

file.open("shortageOfIngredients.txt", ios::in);

while (!file.eof())

{

getline(file, line);

if (line == "")

{

continue;

}

ingredientsInShortage[numberOfIngredientsInShortage] = parse(line, 1);

quantityInShortage[numberOfIngredientsInShortage] = stoi(parse(line, 2));

numberOfIngredientsInShortage++;

}

file.close();

}

// add excess ingredients

void addIngredientsInExcess()

{

string ingredientInExcess, quantityByWhichItWasInExcess;

while (true)

{

system("cls");

header();

subHeader("ingredients in excess");

cout << "Enter ingredient in excess: (Enter 0 to go back)";

cin >> ingredientInExcess;

if (ingredientInExcess == "0")

{

break;

}

if (isIngredientInTheTable(ingredientInExcess))

{

ingredientsInExcess[numberOfIngredientsInExcess] = ingredientInExcess;

}

else

{

system("cls");

cout << "This ingredient is not in the recipe table. Press any key to try again." << endl;

getch();

continue;

}

cout << "Enter quantity by which it was in shortage: ";

cin >> quantityByWhichItWasInExcess;

if (stringNumberValidate(quantityByWhichItWasInExcess))

{

quantityInExcess[numberOfIngredientsInExcess] = stoi(quantityByWhichItWasInExcess);

numberOfIngredientsInExcess++;

addIngredientsInExcessToFile();

}

else

{

system("cls");

cout << "Enter valid quantity. Press any key to try again." << endl;

getch();

continue;

}

}

}

// add excess ingredients to file

void addIngredientsInExcessToFile()

{

fstream file;

file.open("excessOfIngredients.txt", ios::out);

for (int i = 0; i < numberOfIngredientsInExcess; i++)

{

file << ingredientsInExcess[i] << "," << quantityInExcess[i] << endl;

}

file.close();

}

// read excess ingredients from file

void readIngredientsInExcessFromFile()

{

fstream file;

string line;

file.open("excessOfIngredients.txt", ios::in);

while (!file.eof())

{

getline(file, line);

if (line == "")

{

continue;

}

ingredientsInExcess[numberOfIngredientsInShortage] = parse(line, 1);

quantityInExcess[numberOfIngredientsInShortage] = stoi(parse(line, 2));

numberOfIngredientsInExcess++;

}

file.close();

}

// add feedback from employee

void addfeedbackFromEmployee(string employeeName)

{

string paragraphOfFeedback;

string controllerOfControl;

int i = 0;

while (true)

{

system("cls");

header();

subHeader("Feedback");

cout << "Enter your feedback: (Enter 0 to go back)";

getline(cin, paragraphOfFeedback);

if (paragraphOfFeedback == "0")

{

break;

}

if (i != 0)

{

employeeFeedbacks[numberOfFeedbacksAdded] = paragraphOfFeedback;

feedbackEmployeeName[numberOfFeedbacksAdded] = employeeName;

numberOfFeedbacksAdded++; // a glitch i can't figure out. getline tends to take an empty string initially

addFeedbackToFile();

}

i++;

}

}

// view feedback from employee

void viewFeedbackFromEmployee()

{

string serialNumberForDetail;

while (true)

{

system("cls");

header();

subHeader("Employee feedback");

cout << "Serial Number: "

<< "\t\t"

<< "Employee Name"

<< "\t\t"

<< "Feedback" << endl

<< endl;

for (int i = 0; i < numberOfFeedbacksAdded; i++)

{

cout << i + 1 << "\t\t" << feedbackEmployeeName[i] << "\t\t";

for (int j = 0; j < 10; j++)

{

cout << employeeFeedbacks[i][j];

}

cout << "......" << endl;

}

cout << endl;

cout << "Enter serial number for detail view: (Enter 0 to go back)";

cin >> serialNumberForDetail;

if (serialNumberForDetail == "0")

{

break;

}

if (stringNumberValidate(serialNumberForDetail))

{

if (stoi(serialNumberForDetail) > 0 && stoi(serialNumberForDetail) <= numberOfFeedbacksAdded)

{

detailFeedbackView(stoi(serialNumberForDetail) - 1);

}

else

{

cout << "Incorrect number. Press any key to try again" << endl;

getch();

continue;

}

}

else

{

cout << "Incorrect number. Press any key to try again" << endl;

getch();

continue;

}

}

}

// detail view

void detailFeedbackView(int serialNumber)

{

system("cls");

cout << employeeFeedbacks[serialNumber] << endl

<< endl;

cout << "Press any key to continue: " << endl;

getch();

}

// add feedback to file

void addFeedbackToFile()

{

fstream file;

file.open("feedback.txt", ios::out);

for (int i = 0; i < numberOfFeedbacksAdded; i++)

{

file << feedbackEmployeeName[i] << "," << employeeFeedbacks[i] << endl;

}

file.close();

}

// read feedback from file

void readFeedbackFromFile()

{

fstream file;

string line;

file.open("feedback.txt", ios::in);

while (!file.eof())

{

getline(file, line);

if (line == "")

{

continue;

}

feedbackEmployeeName[numberOfFeedbacksAdded] = parse(line, 1);

employeeFeedbacks[numberOfFeedbacksAdded] = parse(line, 2);

numberOfFeedbacksAdded++;

}

file.close();

}

// add recipe ingredients change requests

void addRequests(string employeeUsername)

{

string newIngredient;

while (true)

{

system("cls");

header();

subHeader("Adding ingredients in recipes");

cout << "Enter the name of ingredient that you want to add to the table: (Press 0 to go back)";

cin >> newIngredient;

if (newIngredient == "0")

{

break;

}

if (isIngredientInTheTable(newIngredient))

{

system("cls");

cout << "This ingredient already exists. Press any key to try again." << endl;

getch();

continue;

}

requestedIngredientChangesInRecipes[numberOfRequestsForChangeInRecipes] = newIngredient;

nameOfEmployeesWhoRequestedTheAddition[numberOfRequestsForChangeInRecipes] = employeeUsername;

numberOfRequestsForChangeInRecipes++;

addIngredientChangeRequestToFile();

system("cls");

cout << "Thank you for your request. It will be considered by the admins" << endl;

cout << "Press any key to continue" << endl;

getch();

}

}

// add ingredient change requests to file

void addIngredientChangeRequestToFile()

{

fstream file;

file.open("requestInChangingRecipeIngredients.txt", ios::out);

for (int i = 0; i < numberOfRequestsForChangeInRecipes; i++)

{

file << requestedIngredientChangesInRecipes[i] << "," << nameOfEmployeesWhoRequestedTheAddition[i] << endl;

}

file.close();

}

// reading ingredient change requests from file

void readingIngredientChangeRequestFromFile()

{

fstream file;

string line;

file.open("requestInChangingRecipeIngredients.txt", ios::in);

while (!file.eof())

{

getline(file, line);

if (line == "")

{

continue;

}

requestedIngredientChangesInRecipes[numberOfRequestsForChangeInRecipes] = parse(line, 1);

nameOfEmployeesWhoRequestedTheAddition[numberOfRequestsForChangeInRecipes] = parse(line, 2);

numberOfRequestsForChangeInRecipes++;

}

file.close();

}

// helpful side functions

bool stringNumberValidate(string number) // to check if a string can be correctly evaluated into a number

{

bool flag = true;

for (int i = 0; i < number.length(); i++)

{

if (number[i] < 48 || number[i] > 57)

{

flag = false;

break;

}

}

return flag;

}

string parse(string line, int field)

{

int commaCount = 1;

string requiredString = "";

for (int i = 0; i < line.length(); i++)

{

if (line[i] == ',')

{

commaCount++;

}

else if (commaCount == field)

{

requiredString += line[i];

}

}

return requiredString;

}

int numberOfRequestsByAUser(string usernameOfEmployee)

{

int numberOfRequestsByAUser = 0;

for (int i = 0; i < numberOfRequests; i++)

{

if (ingredientRequestEmployeeName[i] == usernameOfEmployee)

{

numberOfRequestsByAUser++;

}

}

return numberOfRequestsByAUser;

}

int indexOfRequestBySerialNumber(string serialNumberToUpdate, string usernameOfUser)

{

int indexOfRequest = -1;

int requestCount = 0;

for (int i = 0; i < numberOfRequests; i++)

{

if (ingredientRequestEmployeeName[i] == usernameOfUser)

{

requestCount++;

}

if (requestCount == stoi(serialNumberToUpdate))

{

indexOfRequest = requestCount - 1;

}

}

return indexOfRequest;

}

void header()

{

cout << " ##############################" << endl;

cout << "" << endl;

cout << " /$$$$$$ /$$ /$$" << endl;

cout << " /$$\_\_ $$| $$ | $$" << endl;

cout << "| $$ \\\_\_/| $$$$$$$ /$$$$$$ /$$$$$$ /$$$$$$ /$$$$$$ /$$$$$$$" << endl;

cout << "| $$ | $$\_\_ $$ /$$\_\_ $$ /$$\_\_ $$|\_ $$\_/ /$$\_\_ $$ /$$\_\_\_\_\_/" << endl;

cout << "| $$ | $$ \\ $$| $$$$$$$$| $$$$$$$$ | $$ | $$ \\ $$| $$$$$$ " << endl;

cout << "| $$ $$| $$ | $$| $$\_\_\_\_\_/| $$\_\_\_\_\_/ | $$ /$$| $$ | $$ \\\_\_\_\_ $$" << endl;

cout << "| $$$$$$/| $$ | $$| $$$$$$$| $$$$$$$ | $$$$/| $$$$$$/ /$$$$$$$/" << endl;

cout << " \\\_\_\_\_\_\_/ |\_\_/ |\_\_/ \\\_\_\_\_\_\_\_/ \\\_\_\_\_\_\_\_/ \\\_\_\_/ \\\_\_\_\_\_\_/ |\_\_\_\_\_\_\_/ " << endl;

cout << " " << endl;

cout << " ######################################" << endl;

}

void subHeader(string nameMenu)

{

cout << endl;

cout << "----------------------------------------------------------------------------------------------" << endl;

cout << " " << nameMenu << " Menu " << endl;

cout << "----------------------------------------------------------------------------------------------" << endl;

}

* **Weakness in the Business Application**
* The application does not provide the validation for csv format, where everything stored in a file must be validated that they do not contain any commas. This can cause potential issues while loading/reading.
* The requests presented by the employees have been stored in files but there is no proper way to show those requests to either admin or employee.
* getline() was barely used throughout the code, leaving the inputs susceptible to spacing.
* **Future Directions**
* The application has main functionalities set up but there are still many things that the application can do such as calculate cost/profit. Also, by adding a third user of application i.e, the delivery guys, etcetera, we can make the application more complex and interesting. If such concepts are taught to us in the future that could provide an efficient way to handle such complexity, I would definitely continue to make this app better.

**Student Reg. No. :**   **Student Name.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **A-Extensive Evidence** | **B-Convincing Evidence** | **C-Limited Evidence** | **D-No Evidence** |
| Documentation Formatting  **Grade:** | All the documentation meets all the criteria. | Documentation is well formatted but some of the criteria is not fulfilled. | Documentation is required a lot of improvement. | Documentation is not Available |
| **Documentation Formatting Criteria:** In **Binder**, **Title** Page, **Header**-Footers, Font **Style**, Font **Size** all are all consistence and according to given **guidelines**. Project **Poster** is professionally design and well presented | | | | |
| Documentation Contents  **Grade:** | Documentation includes all of the criteria. | Documentation meet more than 80% of the criteria given. | Documentation meet more than 50% of the criteria. | When the documentation meet less than 50% of the criteria. |
| **Documentation Contents Criteria:** **Title** Page - **Table** of Contents - Project **Abstract** - **Functional** Requirements - **Wire** Frames –**Data Flow** Diagram-**Data** Structure (Arrays)-**Function** Headers and Description -Project **Code.** - **Weakness** in the Project and **Future** Directions. - **Conclusion** and What your **Learn** from the Project and Course and What is your **Future** Planning. | | | | |
| Project Complexity  **Grade:** | Project has at least 2 user’s types and each user has at least 5 functionalities. | Project complexity meet 80% criteria given in extensive evidence | Project complexity meet 50% criteria given in extensive evidence | Project complexity meet less than 50% criteria given in extensive evidence |
| Code Style  **Grade:** | All Code style criteria is followed | All code style criteria followed but some improvements required | lot of improvements required in coding style. | **Did not follow** code style, |
| **Code Style Criteria:**  Consistent code style. Code is well indented. Variable and Function names are well defined.  White Spaces are well used. Comments are added. | | | | |
| Code Documentation Mapping  **Grade:** | Code and documentation is synchronized. | Code and documentation does not synchronized at **some** places | Code and documentation does not synchronized at **many** places | Code and documentation **does not** synchronized. |
| Data Structure (Arrays)  **Grade:** | Data structure is sufficient for the project requirements | Data Structure is sufficient but require improvement to meet project requirements. | Data structure is not sufficient and need a lot of improvement | Data Structure is not properly identified and declared. |
| Modularity  **Grade:** | Meet all Modularity criteria | Meet all Modularity criteria but at some places it is missing | Do not sufficiently meet the modularity criteria. | No modularity or very minimum modularity. |
| **Modularity criteria:** Functions are defined for each major feature. Functions are independent (identify from parameter list and return types). | | | | |
| Validations  **Grade:** | Validations on all number type inputs are applied | Validations are applied but at some places it is missing. | Validations are missing at lot of places | No Validations are used |
| File Handling  **Grade:** | Separate files for separate data. Data in csv format | File handing require some improvements | File handing require a lot of improvements | Not implemented |
| Aesthetics of the User Interface  **Grade:** | UI is presentable. Proper coloring, Headers and clear screen is done | UI require some improvements | UI require a lot of improvements | Not implemented |
| Presentation and Demo  **Grade:** | Presentation and Demo was 100% working | Presentation and Demo require some improvements | Presentation and Demo require a lot of improvements | Presentation was not ok and Demo was not working |
| Student Understanding with the Code.  **Grade:** | Student has complete understanding how the code is working and knows the concept. | Student has good understand but some place he does not know the concepts | Student has a very little understand and lack the major concepts. | Student does not have any level of understanding of the code. |

|  |  |
| --- | --- |
| **Checked by:** |  |
| **Comments:** |  |