Sri Lanka Institute of Information Technology

Assignment 2

MLB_WD_04.02_03

Online Recipe Management System

Object Oriented Concepts – IT1050

B.Sc. (Hons) in Information Technology

SLIIT
Discover Your Future

Topic : **Online Recipe Management System**

Group no : **MLB_WD_04.02_03**

Campus : **Malabe**

Submission Date : **19/05/2023**

We declare that this is our own work, and this Assignment does not incorporate without acknowledgment any material previously submitted by anyone else in SLIIT or any other university/Institute. And we declare that each one of us equally contributed to the completion of this Assignment.

Registration no.	Name	Contact no.
IT23227804	N.V. RANDUNUGE	070 370 7141
IT23227354	W.C.S.A LOWE	077 451 5896
IT23218758	T.R.S SURAWEERA	071 726 9223
IT23223912	L.H.R.C LAMAHEWA	076 441 4442
IT23228658	I.W.K HASARANGA	071 527 2946

SLIIT
Discover Your Future

Table of Contents

Description of the Requirements

- In this online recipe management system, a user can register to the system by providing personal details such as Name, Address, Email, Phone no, Gender, and then creating an unique username and a password.
- A Registered user can view recipes and can purchase e-cookbooks to learn more about cooking.
- They can buy one or more than one e-Cookbook at a time by adding them to the cart.
- The system keeps track of sales and generates sales report at the end of the month.
- A Registered user can also upload their own recipes to the system.
- A user-uploaded recipe first needs to be tested by the team of Recipe Testers before it is decided whether to be published on the system or not.
- There are several Recipe Testers working together in the team.
- A member of the Recipe Testing Team should have the necessary qualifications and experience to be a part of the team.
- Once the Recipe testing team tests out the recipe, they send a test report to the admin.
- There are two types of administrators handling this online recipe management system: System administrators and Content administrators.
- System administrators have access to settings and controls that apply across the entire system.
- Content administrator manages content such as Recipes, E-cookbooks, and make decisions on whether to approve or reject the users' recipes depending on the test results.
- Only if the recipe was approved by the content admin, it will be published on the website.
- If users have any issues within the system, they can contact the support agent.
- There are several support agents working for the system, and at least one agent will be available to answer user inquiries at any time.

Classes Identified

- User
- Recipe
- E-Cookbook
- Recipe Testing Team
- Recipe Tester (part of Recipe Testing Team)
- Admin
- System Admin (inherited from Admin class)
- Content Admin (inherited from Admin class)
- Sales
- Support Agent

CRC Cards

User Class	
Responsibilities	**Collaborations**
Register to the system	
Verify User	
Store user'personal information	
Upload new recipes	Recipe class
Purchase E-cookbooks	E-Cookbook class
Edit profile information	
Check FAQ	Support Agent class

Recipe Class	
Responsibilities	**Collaborations**
Store recipe details (name, ingredients, instructions, etc.)	
Delete recipe	Admin class
Update recipe	Admin class

eCookbook Class	
Responsibilities	**Collaborations**
Store E-cookbook details	
Manage E-cookbook content	

Recipe Testing Team	
Responsibilities	**Collaborations**
Store tested recipe details	Recipe class
Test recipes	
Report test results to Admin	Content Admin class

Recipe Tester Class	
Responsibilities	**Collaborations**
Store team member details	Recipe Testing Team class
Update Recipe Tester's details	
Rate recipe quality	Recipe class

Admin Class	
Responsibilities	**Collaborations**
Store administrator's information	
Verify admin login	

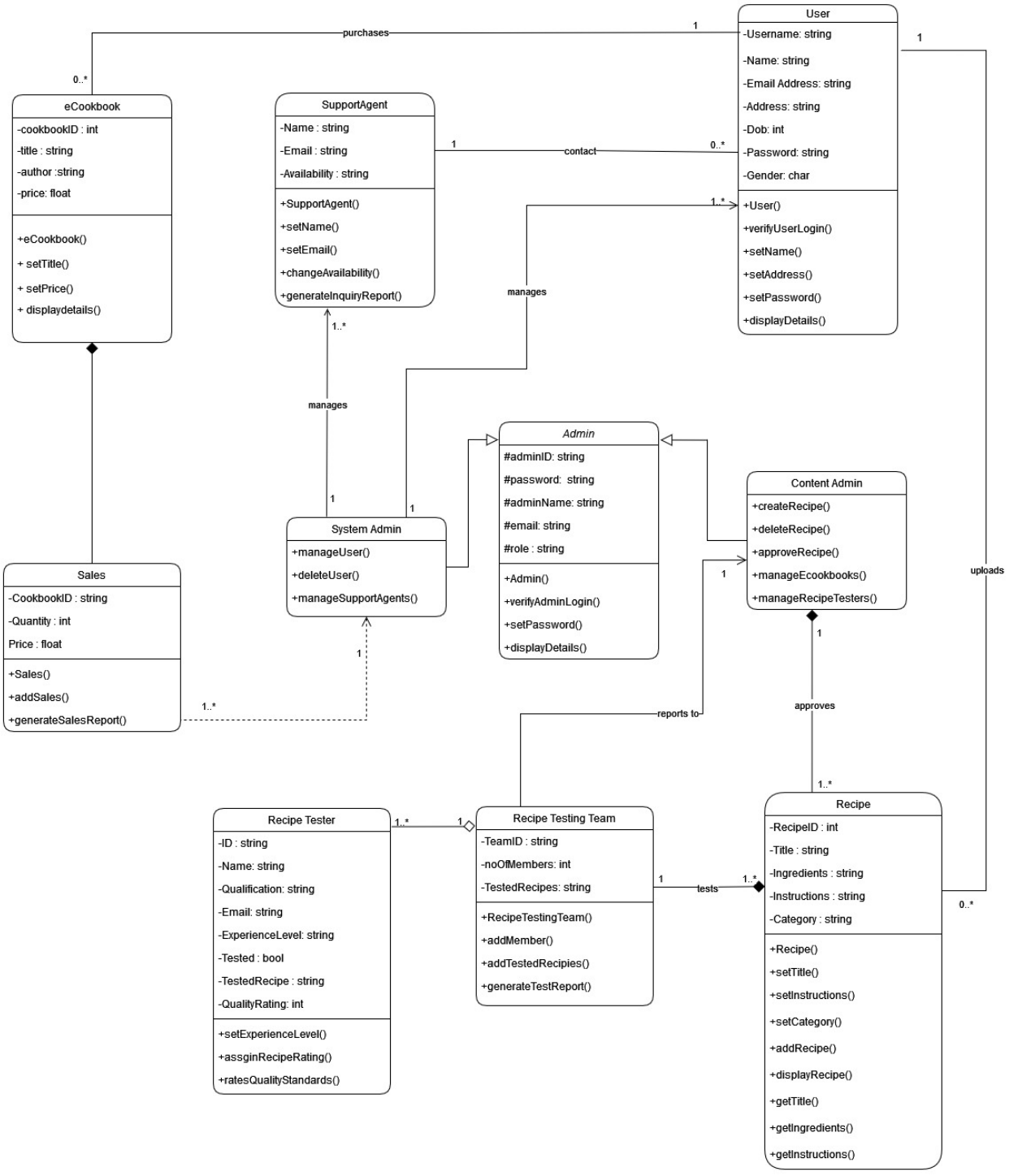
System Admin Class	
Responsibilities	**Collaborations**
Manage User accounts	User class
Manage Support agents	Support Agent class
Manage system-wide settings and controls	
Supervise and approve sales report	Sales Report class

Content Admin Class	
Responsibilities	**Collaborations**
Manage Recipes	Recipe class
Manage approval process of user-uploaded recipes	Recipe Testing Team class, Recipe class
Delete recipes	Recipe class
Manage e-cookbooks	eCookbook class

Support Agent Class	
Responsibilities	**Collaborations**
Store Support Agent's details	
Provide customer support	
Solve user queries and issues	
Generate Inquiry report	
Inform System admin to troubleshoot	

Sales Class	
Responsibilities	**Collaborations**
Store sales information	
Generate sales report	
Provide sales data analysis	System Admin class

Class Diagram



Coding for the Classes

Main.cpp

```
#include "Recipe.h"
#include "Users.h"
#include "eCookbooks.h"
#include "SupportAgent.h"
#include "ContentAdmin.h"
#include "SystemAdmin.h"
#include "Sales.h"
#include "RecipeTestingTeam.h"
#include <iostream>
#include <vector>

using namespace std;

int main() {

    // Declaring array variables

    int dateOfBirth[3] = {2003, 5, 20};
    char passcode[8] = {"j123gf"};

    string ingredients[20] = { "Flour",
                               "Sugar",
                                "Eggs",
                                "Butter",
                                "Milk",
                                "Baking Powder",
                                "Vanilla Extract",
                                "Chocolate Chips"};

    string instructions[50] = {
        "Preheat oven to 350 degrees F (175 degrees C).",
        "Grease and flour a 9x9 inch pan.",
        "In a medium bowl, cream together the sugar and butter.",
        "Beat in the eggs, one at a time, then stir in the vanilla.",
        "Combine flour and baking powder, add to the creamed mixture and mix well.",
        "Finally stir in the milk until batter is smooth.",
        "Pour or spoon batter into the prepared pan."
    };

    string teamMembers[10] = {"Amanda Helani", "Samantha Jones", "Jane Smith"};
    string testedRecipes[100] = {"Vegan Chocolate Cake", "Vegan Strawberry Cake",
"Vegan Scorns"};


    string qualification[20] = {"MSc (Hons) Food Science", "Bsc (Hons) Computer
Science"};
```

```
  // Creating a dynamic object for User class
    User *user1;

    user1 = new User("user01", passcode, "John Doe", "johndoe@gmail.com",
dateOfBirth, "Male", "123 Main St, Anytowne, USA");

    cout << endl << "User Profile" << endl;
    cout << "=============" << endl;
    user1->displaydetails();


    cout << endl << endl;


    // Creating a dynamic object for Recipe class
    Recipe *recipe1;
    recipe1 = new Recipe(1, "Chocolate Cake", ingredients, instructions, "Dessert","1
hour");

    cout << "Recipe Details " << endl;
    cout << "===============" << endl;
    recipe1->displayRecipe();


       cout << endl << endl;


    //Creating an instance for RecipeTestingTeam class
    RecipeTestingTeam Team1("TEAM001", "Vegan Recipe Developers' Team", 3,
teamMembers, testedRecipes);

    Team1.addMember("Manisha Henric");
    Team1.addTestedRecipe("Scalion Biscuits");

    Team1.displayTeamDetails();


    cout << endl << endl;


    // Creating an object for RecipeTester class
    RecipeTester *tester1;
    tester1 = new RecipeTester("TS001", "TEAM001", "Jane Smith", "jane@gmail.com",
qualification, "Professional-Level", "Vegan Chocolate Cake", true, 5);

    cout << "Recipe Tester Details" << endl;
    cout << "=====================" << endl << endl;
    tester1->displayDetails();
```

```
   // Creating a dynamic object for ContenetAdmin class
    ContentAdmin *cAdmin1;
    cAdmin1 = new ContentAdmin("CA001", "123pwd", "Rebecca Black",
"rebecca.b@gmail.com", "Content Admin");


    cout << endl << endl;


    // Creating a dynamic object for SystemAdmin class
    SystemAdmin *sAdmin1;
    sAdmin1 = new SystemAdmin("SA001", "12wd45", "Will Smith", "will@gmail.com" ,
"System Admin");


    cout << "Administrators' Details" << endl;
    cout << "======================" << endl;
    sAdmin1 -> displayDetails();  // display System Admin details
    cAdmin1 -> displayDetails();  // display Content Admin details

    cout << endl << endl;


    // Creating an eCookbook object
    eCookbook ecookbook1 ("CK001", "The Greenhouse Cookbook", "Emma Knight", 1700.0);

    cout << "E-Cookbook Details" << endl;
    cout << "==================" << endl;
    ecookbook1.displaydetails();


    cout << endl << endl;

    // Create an object for Sale class
    Sale *salesreport;
    salesreport = new Sale("CK001", 15, 1700.0);

    salesreport -> generateSalesReport();


    cout << endl << endl << endl;

    // Create an object for SupportAgent
    SupportAgent *Agent1;
    Agent1 = new SupportAgent("Alice Stones", "alice@gmail.com", "Available");

    // Display Inquiry Report
    cout << "Enter inquiry details to gernaerte Inquiry Report >>" << endl << endl;
    Agent1->generateReport();

    return 0;
}
```

Users.h

```
#pragma once
#include <iostream>
using namespace std;

class User
{
    private:
        string username;
        char password[8];
        string name;
        string email;
        int dateOfBirth[3];
        string gender;
        string address;

    public:

         // Constructor
        User(string userName, char uPass[], string uName, string uEmail, int dob[],
string uGen, string uAddress);

        bool UserLogin(string userName, char uPass[]);
        int calcAge(int dob[]);

        void setName(string uName);
        void setEmail(string uEmail);
        void setPassword(char uPass[]);
        void setAddress(string uAddress);
        void setDob(int dob[]);

        string getName();
        string getEmail();
        int getAge();
        int getDob();
        void displaydetails();

        ~User();
};
```

Users.cpp

```
#include "Users.h"
#include <cstring>
#include <ctime>
#include <iostream>

using namespace std;

// Implementation of User Constructor
User::User(string userName, char uPass[], string uName, string uEmail,
           int dob[], string uGen, string uAddress) {
  username = userName;
  strcpy(password, uPass);
  name = uName;
  email = uEmail;

  dateOfBirth[0] = dob[0];
  dateOfBirth[1] = dob[1];
  dateOfBirth[2] = dob[2];

  gender = uGen;
  address = uAddress;
}

bool User::UserLogin(string userName, char uPass[]) {

  return username == userName && strcmp(password, uPass) == 1;
}

// Setter methods

void User::setName(string uName) { name = uName; }

void User::setPassword(char uPass[]) { strcpy(password, uPass); }

void User::setAddress(string uAddress) { address = uAddress; }

void User::setEmail(string uEmail) { email = uEmail; }

void User::setDob(int dob[]) {
  dateOfBirth[0] = dob[0];
  dateOfBirth[1] = dob[1];
  dateOfBirth[2] = dob[2];
}
```

```
// calculate and return age of the user

int User::calcAge(int dob[]) {

 // Get the current time
  time_t now = time(0);
  tm *ltm = localtime(&now);

 // Current date components
  int currentYear = 1900 + ltm->tm_year;
  int currentMonth = 1 + ltm->tm_mon;
  int currentDay = ltm->tm_mday;

 // User's birth date components
  int birthYear = dob[0];
  int birthMonth = dob[1];
  int birthDay = dob[2];

  // Calculate age
  int age = currentYear - birthYear;
  if (currentMonth < birthMonth ||
      (currentMonth == birthMonth && currentDay < birthDay)) {
    age--;
  }

  return age;

  /*Reference to the code segment of getting the current date and time:
  https://www.javatpoint.com/cpp-date-and-time */
}

// Getter methods

string User::getName(){
  cout << name << endl;
  return "";
}

string User::getEmail(){
  cout <<  email << endl;
  return "";
}

int User::getAge() {

  int calAge(int dob[]);
  return 0;
}
```

```
int User::getDob()  {
  int year = dateOfBirth[0];
  int month = dateOfBirth[1];
  int day = dateOfBirth[2];

  cout << year << "/" << month << "/" ;

  return day;
}

// Display user information

void User::displaydetails() {
  cout << "User ID: " << username << endl;
  cout << "Name: " << name << endl;
  cout << "Email: " << email << endl;
  cout << "Gender: " << gender << endl;
  cout << "Address: " << address << endl;
  cout << "Age: " << calcAge(dateOfBirth) << endl;
  cout << "Date of Birth: " << getDob() << endl;
}

User::~User() {

}
```

Recipe.h

```
#pragma once
#include <iostream>
#include <string.h>
using namespace std;

class Recipe
{
	private:
	  int RecipeID;
	  string Title;
	  string Ingredients[20];
	  string Instructions[50];
	  string Category;
	  string time;

	public:
	  //Defining a constructor with parameters
	  Recipe(int rID, string rTitle, string Ingredients[], string
	Instructions[],string Category, string time);

	  void setTitle(string rTitle);
	  void setIngredients(string rIngredients[]);
	  void setInstructions(string rInstructions[]);
	  void setTime(string rTime);

	  void displayRecipe();
	  string getTitle();
	  string getIngredients();
	  string getInstructions();
};
```

Recipe.cpp

```
#include "Recipe.h"
#include <iostream>

using namespace std;

// Implementation of Recipe Constructor

Recipe::Recipe(int rID, string rTitle, string rIngredients[], string rInstructions[],
string rCategory, string rTime) {

    RecipeID = rID;
    Title = rTitle;

    for (int i = 0; i < 20; i++) {
        Ingredients[i] = rIngredients[i];
    }

    for (int i = 0; i < 50; i++) {
        Instructions[i] = rInstructions[i];
    }

    Category = rCategory;
    time = rTime;
}

// Implementation of set methods

void Recipe::setTitle(string rTitle) { Title = rTitle; }

void Recipe::setTime(string rTime) { time = rTime; }

void Recipe::setIngredients(string *rIngredients) {
    for (int i = 0; i < 20; ++i) {
        Ingredients[i] = rIngredients[i];
    }
}

void Recipe::setInstructions(string *rInstructions) {
    for (int i = 0; i < 50; ++i) {
        Instructions[i] = rInstructions[i];
    }
}


string Recipe::getTitle() { return Title; }
```

```
string Recipe::getIngredients() {
    for (int i = 0; i < 20; ++i) {
        if (!Ingredients[i].empty()) {
          cout << Ingredients[i] << ", ";
        }
      }
  return "";
}

string Recipe::getInstructions() {
    for (int i = 0; i < 50; ++i) {
        if (!Instructions[i].empty()) {
          cout << i + 1 << ". " << Instructions[i] << endl;
        }
    }
  return "";
}

// Implementation of dispalyRecipe function

void Recipe::displayRecipe() {
  cout << "Recipe ID: " << RecipeID << endl;
  cout << "Recipe Name: " << Title << endl << endl;
  cout << "Ingredients: " << getIngredients() << endl << endl;
  cout << "Instructions: " << endl;
  getInstructions();

  cout << endl <<"Category: " << Category << endl;
  cout << "Time to Make: " << time << endl;
}
```

eCookbooks.h

```
#pragma once
#include <iostream>
using namespace std;

class eCookbook
{
	private:
	    int cookbookID;
	    string title;
	    string author;
	    float price;

	public:
	    eCookbook(int bookID, string bookTitle, string bookAuthor, float
	bookPrice);
	    void setTitle(string bookTitle);
	    void setAuthor(string bookAuthor);
	    void setPrice(float bookPrice);
	    void displaydetails();

	    string getTitle();
	    string getAuthor();
	    float getPrice();
};
```

eCookbooks.cpp

```
#include "eCookbooks.h"
#include <iostream>
#include <iomanip>
using namespace std;

//Constructor

eCookbook::eCookbook(int bookID, string bookTitle, string bookAuthor, float
bookPrice){
    cookbookID = bookID;
    title = bookTitle;
    author = bookAuthor;
    price = bookPrice;
}


//set methods

void eCookbook::setTitle(string bookTitle) {
    title = bookTitle;
}

void eCookbook::setAuthor(const string bookAuthor) {
    author = bookAuthor;
}

void eCookbook::setPrice(float bookPrice) {
    price = bookPrice;
}

//Getter methods

string eCookbook::getTitle() {
    return title;
}

string eCookbook::getAuthor() {
    return author;
}

float eCookbook::getPrice() {
    return price;
}
//display eCokbook details

void eCookbook::displaydetails() {
    cout << "Cookbook ID: " << cookbookID << std::endl;
    cout << "Title: " << title << endl;
    cout << "Author: " << author << endl;
    cout << "Price: Rs." << std::fixed << std::setprecision(2) << price
<< endl;
}
```

SupportAgents.h

```
#pragma once
#include <iostream>
using namespace std;

class SupportAgent
{
    private:
        string name;
        string email;
        string availability;


    public:
        SupportAgent(string sName, string sMail, string sAvailability); //constructor
        void changeAvailability(string sAvb);
        void generateReport();
        ~SupportAgent();    //destructor
};
```

SupportAgents.cpp

```
#include <iostream>
#include <string>
#include "SupportAgent.h"
using namespace std;

//Implementing constructor
SupportAgent::SupportAgent(string sName, string sMail, string sAvailability)
{
    name = sName;
    email = sMail;
    availability = sAvailability;
}

void SupportAgent::changeAvailability(string sAvb)
{
    availability = sAvb;
}



void SupportAgent::generateReport()
{
    string RepID, cusName, cusEmail, inqID, inqDate, inqChannel, inqSubject, inqDesc,
Prio, actions;

    cout << "Enter Report ID: ";
    getline (cin, RepID);
    cout << "Enter Customer Name: ";
    getline (cin, cusName);
    cout << "Enter Customer Email: ";
    getline (cin, cusEmail);
    cout << "Enter Inquiry ID: ";
    getline (cin, inqID);
    cout << "Enter Inquiry Date: ";
    getline (cin, inqDate);
    cout << "Enter Inquiry Channel: ";
    getline (cin, inqChannel);
    cout << "Enter Inquiry Subject: ";
    getline (cin, inqSubject);
    cout << "Enter Inquiry Description: ";
    getline (cin, inqDesc);
    cout << "Enter Priority: ";
    getline (cin, Prio);
    cout << "Enter Actions: ";
    getline (cin, actions);

    cout << endl << endl << endl;
```

```
    cout << endl<< "Customer Inquiry Report" << endl;
    cout << "========================" << endl << endl;

    cout << "Report ID: " << RepID<< endl;
    cout << "Customer Name: " << cusName << endl;
    cout << "Contact Information: " << cusEmail << endl << endl;

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

    cout << "Inquiry ID: " << inqID << endl<< endl;
    cout << "Date of Inquiry: " << inqDate << endl;
    cout << "Inquiry Channel: " << inqChannel << endl;
    cout << "Subject of Inquiry: " << inqSubject << endl;
    cout << "Description of Inquiry: " << inqDesc << endl;
    cout << "Priority Level: " << Prio << endl;
    cout << "Actions Taken: " << actions << endl << endl;

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


    cout << "Support Agent Signature: " << name << endl;


    // Implementing the destructor

     SupportAgent::~SupportAgent(){

     }

}
```

Admin.h

```
#pragma once
#include <iostream>
using namespace std;

#include <string>

class Admin {
      protected:
          string adminID;
          string password;
          string adminName;
          string email;
          string role;

      public:
          void verifyAdminLogin();
          void setPassword(const string& pwd);
          void setEmail(const string& mail);
          void setRole(const string& rl);
          void setAdminName(const string& name);
};
```

Admin.cpp

```
#include "Admin.h"
#include <iostream>

using namespace std;

void Admin::verifyAdminLogin() {

    string username, pwd;

    cout << "Enter username : " ;
    cin >> username;
    cout << "Enter password : " ;
    cin >> pwd;

    // check if the entered credentials match the admin credentials
    if (username == adminID && pwd == password) {
        cout << "Login Succesful!" << endl;

    } else {
        cout << "Login Failed!" << endl;
    }
}


void Admin::setPassword(const string& pwd) {
    password = pwd;
}

void Admin::setEmail(const string& mail){
    email = mail;
}

void Admin::setRole(const string& rl){
    role = rl;
}

void Admin::setAdminName(const string& name){
    adminName = name;
}
```

SystemAdmin.h

```
#pragma once
#include <iostream>
#include "Admin.h"

using namespace std;

class SystemAdmin : public Admin     // This class is a sub class of the Admin class
{
    public:
        SystemAdmin(string id,  string pwd,  string name,  string mail,  string rl);
        void displayDetails();
        void manageUser();
        void deleteUser();
        void manageSupportAgents();
};
```

SystemAdmin.cpp

```
#include "SystemAdmin.h"
#include <iostream>

using namespace std;

SystemAdmin::SystemAdmin(string id, string pwd, string name, string mail, string rl)
{
    adminID = id;
    password = pwd;
    adminName = name;
    email = mail;
    role = rl;
}

void SystemAdmin::displayDetails(){
    cout << "Admin ID: " << adminID << endl;
    cout << "Admin Name: " << adminName << endl;
    cout << "Email: " << email << endl;
    cout << "Role: " << role << endl << endl;


}
void SystemAdmin::manageUser() {

}

void SystemAdmin::deleteUser() {

}

void SystemAdmin::manageSupportAgents() {

}
```

ContentAdmin.h

```
#pragma once
#include <iostream>
#include "Admin.h"

using namespace std;

class ContentAdmin : public Admin     // This class is a sub class of the Admin class
{

    public:

        ContentAdmin(string cid, const string& cpwd, const string& cname, const
string& cmail, const string& crole);

        void displayDetails();

        void createRecipe();
        void deleteRecipe();
        void updateRecipe();
        bool approveRecipe();
        void manageEcookbooks();
        void manageRecipeTestingTeams();
};
```

ContentAdmin.cpp

```
#include <iostream>
#include "ContentAdmin.h"
using namespace std;

// Implementation of the class constructor
ContentAdmin::ContentAdmin(string cid, const string& cpwd, const string& cname, const
string& cmail, const string& crole){

   adminID = cid;
   password = cpwd;
   adminName = cname;
   email = cmail;
   role = crole;
}

void ContentAdmin::displayDetails(){

    cout << "Admin ID: " << adminID << endl;
    cout << "Admin Name: " << adminName << endl;
    cout << "Email: " << email << endl;
    cout << "Role: " << role << endl;


}

bool ContentAdmin::approveRecipe(){
 return true;
}

void ContentAdmin::createRecipe(){

}

void ContentAdmin::deleteRecipe(){

}

void ContentAdmin::updateRecipe(){

}
void ContentAdmin::manageEcookbooks(){

}
void ContentAdmin::manageRecipeTestingTeams(){

}
```

RecipeTestingTeam.h

```
#pragma once
#include <iostream>

using namespace std;

class RecipeTestingTeam {
   private:
      string teamID;
      string teamName;
      int noOfMembers;
      string teamMembers[10];
      string testedRecipes[100];

  public:
      RecipeTestingTeam(string id, string name, int noMembers, string members[],
string recipes[]);
      void addMember(string member);
      void removeMember(string member[]);
      void addTestedRecipe(string recipes);
      void generateTestReport();

      void displayTeamDetails();
      ~RecipeTestingTeam();
};
```

RecipeTestingTeam.cpp

```
#include <iostream>
#include "RecipeTestingTeam.h"

using namespace std;

// Implementing the class constructor

RecipeTestingTeam::RecipeTestingTeam(string id, string name, int noMembers, string
members[], string recipes[])
{
  teamID = id;
  teamName = name;
  noOfMembers = noMembers;

  for (int i = 0; i < 10; i++) {
    teamMembers[i] = members[i];
  }

  for (int i = 0; i < 100; i++) {
      testedRecipes[i] = recipes[i];
  }

}

// Implementing the addMember() method
void RecipeTestingTeam::addMember(string member){
  if (noOfMembers < 10) {

    teamMembers[noOfMembers++] = member;

  } else {
    cout << "Team is full!" << endl;
  }
}


// Implementing the addTestedRecipe() function to add recently tested recipes
void RecipeTestingTeam::addTestedRecipe(string recipes){

  for (int i = 0; i < 100; ++i) {
      if (testedRecipes[i].empty()) {
          testedRecipes[i] = recipes;
          break;
      }
  }

}
```

```
// Implementing the displayTeamDetails() method
void RecipeTestingTeam::displayTeamDetails(){

  cout << "Recipe Testing Team Infromation "  << endl;
  cout << "================================"  << endl;

  cout << "Team ID: " << teamID << endl;
  cout << "Team Name: " << teamName << endl;
  cout << "No of Members: " << noOfMembers << endl;

  cout << "Team Members: " << endl;
  for (int i = 0; i < 10; ++i) {
    if (!teamMembers[i].empty()) {
      cout << "   " << teamMembers[i] << endl ;
    }
  }

  cout << endl << "Recipes Tested: " << endl;
  for (int i = 0; i < 10; ++i) {
    if (!testedRecipes[i].empty()) {
      cout << i + 1 << ". " << testedRecipes[i] << endl;
    }
  }


}

void RecipeTestingTeam::removeMember(string member[]){


}

void RecipeTestingTeam::generateTestReport(){


}

RecipeTestingTeam::~RecipeTestingTeam(){

}
```

RecipeTester.h

```
#pragma once
#include <iostream>
#include <string>

using namespace std;

class RecipeTester {
private:
    string testerID;
    string teamID;
    string testerName;
    string testerEmail;
    string Qualifications[20];
    string experienceLevel;

    bool tested;
    string testedRecipe;
    int qualityRating;

public:
    // Creating a constructor to initialize values to Recipe Tester objects
    RecipeTester(string tID, string teamid, string tName, string tEmail, string
tQualifications[], string experience, string testedRep, bool Rtested, int rating);

    void displayDetails();
    bool ratesQualityStandards();
};
```

RecipeTester.cpp

```
#include <iostream>
#include "RecipeTester.h"

using namespace std;

RecipeTester::RecipeTester(string tID, string teamid, string tName, string tEmail,
string tQualifications[], string experience, string testedRep, bool Rtested, int
rating){

  testerID = tID;
  teamID = teamid;
  testerName = tName;
  testerEmail = tEmail;

  for (int i = 0; i < 20; i++) {
      Qualifications[i] = tQualifications[i];
  }

  experienceLevel = experience;
  testedRecipe = testedRep;
  tested = Rtested;
  qualityRating = rating;

}

void RecipeTester::displayDetails(){
  cout << "Recipe Tester's personal info :" << endl;
  cout << "------------------------------" << endl;

  cout << "Tester ID: " << testerID << endl;
  cout << "Team ID: " << teamID << endl;
  cout << "Tester's Name: " << testerName << endl;
  cout << "Tester's Email: " << testerEmail << endl;
  cout << "Qualifications: " << endl;

  for (int i = 0; i < 20; ++i) {
    if (!Qualifications[i].empty()) {
      cout << "    " << Qualifications[i] << endl;
    }
  }

  cout << "Experience Level: " << experienceLevel <<endl <<endl <<endl;

  cout << "Rating on Tested Recipe :" << endl;
  cout << "--------------------" << endl;
  cout << "Tested Recipe: " << testedRecipe << endl;
  cout << "Tested: " << tested << endl;
  cout << "Quality Rating: " << qualityRating << endl;

}
```

```
bool RecipeTester::ratesQualityStandards(){
  const int requiredQualityRating = 4;
  // Assuming a quality rating of 4 or higher is needed for approval

  return qualityRating >= requiredQualityRating;

}
```

Sales.h

```
#pragma once
#include <iostream>

using namespace std;

class Sale {

    private:

        string cookbookID;
        int quantity;
        float price;

    public:

        Sale(string ckID, int qty, float prc);
        void addSale(string ckID, int qty, float prc);
        void generateSalesReport();
};
```

Sales.cpp

```
#include <iostream>
#include "Sales.h"
#include <string>
#include <iomanip>

using namespace std;

// Implementing the Constructor

Sale::Sale(string ckID, int qty, float prc)
{
     cookbookID = ckID;
     quantity = qty;
     price = prc;

}

void Sale::addSale(string ckID, int qty, float prc)
{

}

void Sale::generateSalesReport() {

  double totalRevenue = 0;
  int totalItemsSold = 0;

  cout << "Sales Report" << endl;
  cout << "============" << endl;
  cout <<  "Cookbook ID"
       << setw(15) << "Quantity"
       << setw(10) << "Price"
       << setw(10) << "Total" << endl;


  double total = quantity * price;
  totalRevenue += total;
  totalItemsSold += quantity;
  cout <<  cookbookID
       << setw(15) << quantity
       << setw(18) << price
       << setw(11) << total << endl;

  cout << endl << "Total Items Sold: " << totalItemsSold << endl;
  cout << "Total Revenue: Rs." << fixed << setprecision(2) << totalRevenue << endl;
}
```

Individual Contribution	
IT number	**Classes worked on**
IT23227804	RecipeTestingTeam class, RecipeTester class
IT23227354	User class, SystemAdmin class
IT23218758	eCookbook class, Sales class
IT23223912	Recipe class, ContentAdmin class
IT23228658	SupportAgent class, Admin class