
Recipe Management App

Documentation



Indian Institute of Information Technology,
Design & Manufacturing, Kancheepuram

PREPARED BY

Rohit Kumar - CS23B2053

Gyan Chandra - CS23I1053

1 Project Overview

This section provides an overview of the [Recipe Management App](#), which is developed as part of this project using core [Object-Oriented Programming \(OOP\)](#) principles.

1.1 Objective

The primary objective of this project is to create a user-friendly [Recipe Management Application](#) that allows users to store, manage, and retrieve recipe information efficiently. This application leverages OOP concepts such as [encapsulation](#), [abstraction](#), [classes](#), and [composition](#) to ensure modular and maintainable code.

1.2 Features Implemented

In this project, the following OOP features and techniques have been applied:

- **Encapsulation:** Encapsulation involves bundling data (attributes) and functions (behaviours) within a class, ensuring that only the necessary details are accessible to the outside. Each [class](#) represents a real-world entity, such as a recipe or a user, and controls access to its data to maintain integrity and modularity.
- **Abstraction:** Abstraction is the concept of exposing only essential attributes and behaviors of an object while hiding its internal implementation details. By keeping data members [private](#) and providing access through [public member functions](#), abstraction simplifies interaction with the class and helps protect data integrity.
- **Classes and Composition:** [Classes](#) are defined to represent various entities, such as Recipe, User, and Time. Composition is used to show a [has-a](#) relationship.
- **Function Templates:** To improve flexibility, function templates are implemented for sorting/-filtering recipes based on various attributes (such as [likes](#) and [ratings](#)). These templates enable the use of [generic code](#) for different data types and operations, enhancing code reusability.
- **File Handling Implementation:** File handling is a crucial feature implemented in this project to allow users to save and load recipe data. By storing data in text files, the application provides persistent storage for recipes, users, and other relevant information.
- **User Authentication and Authorization:** User authentication and authorization are implemented to secure the Recipe Management App. Users must log in with a valid username and password to access their accounts and manage recipes.



2 Encapsulation-Abstraction

```
class Recipe
{
    friend bool operator==(const Recipe &r1, const Recipe &r2);
    friend ostream &operator<<(ostream &os, const Recipe &r);

private:
    string name;
    vector<pair<string, string> > ingredients;
    string instructions;
    Time prepTime;

    int ratingCount;
    string addcody;

public:
    int likes;
    double ratings;

    Recipe();
    Recipe(string, vector<pair<string, string> >, string, int, int, User, int, int, int);
    Recipe(const Recipe &r);

    void addRatings(double);
    void addIngredients(const string &s, const string &i);
    void removeIngredients(const string &i);
    void increaseLikes();
    string getRecipeName() const;
    string getAddcody() const;
};
```

(a) Recipe Class

```
class User
{
public:
    friend istream &operator>>(istream &is, User &u);
    friend ostream &operator<<(ostream &os, const User &u);

private:
    string username;
    string password;

public:
    User(string = "alice", string = "alice@45");
    string getUsername() const;
    string getPass() const;
    void setUserToNull();
    void setPass(const string &s);
    void setName(const string &s);
};
```

(b) User Class

```
class Time
{
private:
    friend ostream &operator<<(ostream &os, const Time &t);

    int hour;
    int minutes;
    int seconds;

public:
    Time(int = 0, int = 30, int = 0);
};
```

(c) Time Class

Figure 1: Interface of Recipe, User, and Time Classes

The **Recipe** class demonstrates **encapsulation** by making key data members **private**, allowing access only through public **member functions** like **addRatings**, **addIngredients**, and **getRecipeName**. This ensures controlled data access and modification.

It also illustrates **abstraction** by exposing essential behaviors, such as managing ingredients and ratings, while hiding **internal implementation details**. This clean interface simplifies use, enabling interaction without needing to know the internal structure.

The **User** class demonstrates **encapsulation** by keeping sensitive data members, like **username** and **password**, private. Access to these members is controlled through public member functions such as **getUsername**, **getPass**, **setPass**, and **setName**, which allow safe and regulated interaction with the user's information.

The class also provides a clean and simple interface through **abstraction**, exposing essential functionalities (e.g., setting and retrieving username and password) without revealing internal data handling, thus ensuring security and simplicity for users of the class.

The **Time** class demonstrates **encapsulation** by keeping its data members, such as **hour**, **minute**, and **second**, private and accessible only through public methods.

In this context, the **Time** class is used to implement **class composition**. Specifically, an object of the **Time** class is included as a data member of the **Recipe** class to represent the preparation time.

3 User Authentication/Authorization

This part of project implements a basic **user authentication system** with functionalities for **signup, login, and duplicate account prevention**. Below are the key features:

1. Signup:

- User selects the **Signup** option and enters a **username** and **password**.
- Credentials are **saved to a file**.
- A welcome message confirms a successful signup.

2. Login:

- User selects the **Login** option and inputs their credentials.
- If the details match the saved data, the system displays **“Login successful!”**.

3. Duplicate Check:

- If the user tries to sign up with an existing username, the system prevents duplicate accounts.
- Displays a message: **“User already exists! Please Login.”**

```
Enter 1 to Signup into our app
Enter 2 to login
Enter 3 to exit
1
Enter details(username,password)
MyName
Mypassword
User details have been saved to file.
Hello MyName Welcome! to our Recipe App
```

(a) Signup

```
Enter details(username,password)
MyName
Mypassword
User already Exists! Please Login.
```

(b) Duplicate Signup

```
Enter 1 to Signup into our app
Enter 2 to login
Enter 3 to exit
2
Enter username: MyName
Enter password: Mypassword
Login successful!Welcome back MyName
```

(c) Login

Figure 2: Outputs of Authentication

4 Definitions(Authentication)

```
void signUp(User &u)
{
    cout << "Enter details(username,password)" << endl;
    cin >> u;
    ofstream out("user.txt", ios::app);
    if (login(u.getUsername(), u.getPass()))
    {
        cout << "User already Exists! Please Login." << endl;
        u.setUserToNull();
        return;
    }
    if (!out)
    {
        cerr << "Error while opening file for saving user data." << endl;
        return;
    }
    out << u << endl;
    out.close();
    cout << endl;
    cout << "User details have been saved to file." << endl;
    cout << endl;
}
```

(a) Signup Function

```
int handleLogin(User &u)
{
    string name, password;
    cout << "Enter username: ";
    cin >> name;
    cout << "Enter password: ";
    cin >> password;
    cout << endl;

    if (login(name, password))
    {
        cout << "Login successful!" << "Welcome back " << name << endl;
        << endl;
        u.setName(name);
        u.setPass(password);
        return 1;
    }
    else
    {
        cout << "Invalid username or password." << endl;
        << endl;
        return 0;
    }
}
```

(b) Handle login

```
int handleSignup(User &u)
{
    signUp(u);
    if (u.getUsername() == " ")
    {
        return 1;
    }
    cout << "Hello " << u.getUsername() << " Welcome! to our Recipe App" << endl;
    return 0;
}
```

(c) Handle Signup

```
// login function
bool login(const string &username, const string &password)
{
    ifstream in("user.txt");
    if (!in)
    {
        cerr << "Error opening file for reading." << endl;
        return false;
    }
    User u;
    while (in >> u)
    {
        if (u.getUsername() == username && u.getPass() == password)
        {
            return true;
        }
    }
    return false;
}
```

(d) Handle Login

Figure 3: Handlers Functions



5 Function Templates

Function Templates is used as generic programming practice. we can write a generic function that can be used for different data types. Examples of function templates are `sort()`, `max()`, `min()`, `printArray()`.

1. **Template Flexibility:** The `bubbleSortRecipes` template function sorts `Recipe` objects by any specified member (e.g., `likes` or `ratings`) where `likes` is of `int` type while `ratings` is of `double` data type.
2. **Bubble Sort Optimization:** The use of a swapped flag in the bubble sort stops sorting early if the list is already sorted, improving efficiency.
3. **Descending Order Sorting:** The code correctly sorts recipes in **descending order** for both `likes` and `ratings`, as shown in the output.
4. **User Prompt:** The program lets the user choose the sorting criterion, enhancing usability.

```
Enter 1 to filter based on likes(decreasing order)
Enter 2 to filter based on ratings(decreasing order)
2
Sorted by ratings:
Grilled Cheese Sandwich : 5
Pancakes : 4
Overnight Oats : 3
```

(a) Sorting by ratings

```
Enter 1 to filter based on likes(decreasing order)
Enter 2 to filter based on ratings(decreasing order)
1
Sorted by likes:
Grilled Cheese Sandwich : 10
Overnight Oats : 7
Pancakes : 5
```

(b) Sorting by likes

```
template <typename T>
void bubbleSortRecipes(vector<Recipe> &recipes, T Recipe::*member)
{
    bool swapped;
    for (int i = 0; i < recipes.size() - 1; ++i)
    {
        swapped = false;
        for (int j = 0; j < recipes.size() - i - 1; ++j)
        {
            if (recipes[j].*member < recipes[j + 1].*member)
            {
                swap(recipes[j], recipes[j + 1]);
                swapped = true;
            }
        }
        if (!swapped)
            break;
    }
}
```

(c) Code

Figure 4: Function Templates



6 Constructors Definitions

```
Time::Time(int hour, int minutes, int seconds)
{
    if ((hour < 0 || hour > 23) || (minutes < 0 || minutes > 59) || (seconds < 0 || seconds > 59))
    {
        this->hour = 0;
        this->minutes = 30;
        this->seconds = 0;
    }
    else
    {
        this->hour = hour;
        this->minutes = minutes;
        this->seconds = seconds;
    }
}
```

(a) Time Constructor

```
// Constructor definition
Recipe::Recipe(string name, vector<pair<string, string>> ingredients, string instructions,
               int likes, int ratings, User user,
               int hour, int minutes, int seconds)
: prep_time(hour, minutes, seconds)
{
    this->name = name;
    this->ingredients = ingredients;
    this->instructions = instructions;
    this->likes = likes;
    this->ratings = ratings;
    this->addedBy = user.getUsername();
    ratingCount = 1;
}
```

(b) Recipe Constructor

```
Recipe::Recipe(const Recipe &other)
{
    name = other.name;
    ingredients = other.ingredients;
    instructions = other.instructions;
    prep_time = other.prep_time;
    likes = other.likes;
    ratings = other.ratings;
    ratingCount = other.ratingCount;
    addedBy = other.addedBy;
}
```

(c) Recipe Copy Constructor

```
User::User(string username, string pass)
{
    this->username = username;
    this->password = pass;
}
```

(d) User Constructor

Figure 5: Constructors Overloading



7 Operator and Function Overloading

Operator Overloading is the process of giving special meaning to an existing operator in the context of user-defined data types.

Function Overloading is defined as when two or more functions have the same name but differ in their signature (type, number of parameters).

1. **Overloaded < < Operator for Output:**

- This operator is used to **display recipe details** in a well-formatted way.

2. **Overloaded > > Operator for Input:**

- This operator is used to take **inputs** of data members of objects in one go.

3. **Overloaded == Operator for Comparing Recipes:**

- This operator checks for **common ingredients** between two recipes.
- When you use `recipe1 == recipe2;`, it:
 - Prints out the ingredients that both recipes share.
 - Returns **true** if there are common ingredients, otherwise false.

4. **Function Overloading for Recipe:**

- The Recipe constructor is overloaded where the parameterized constructor and copy constructor have been used.

```
Recipe() {};  
Recipe(string, vector<pair<string, string>>, string, int, int, User, int, int, int);  
Recipe(const Recipe &);
```

(a) Constructor Overloading Example

Figure 6: Function Overloading in Recipe Class



8 Overloading Definitions

```
ostream &operator<<(ostream &out, const Recipe &r)
{
    out << "-----" << endl;
    out << "Recipe Details" << endl;
    out << "-----" << endl;
    out << "Recipe Name: " << r.name << endl;
    out << "-----" << endl;

    out << "Ingredients:" << endl;
    out << "-----" << endl;
    out << "Name" << "Quantity" << endl;
    out << "-----" << endl;
    for (int i = 0; i < r.ingredients.size(); i++)
    {
        out << r.ingredients[i].first << " " << r.ingredients[i].second << endl;
    }
    out << "-----" << endl;

    out << "Preparation Time:" << endl;
    out << "-----" << endl;
    out << r.prep_time << endl;
    out << "-----" << endl;

    out << "Cooking Instructions:" << endl;
    out << "-----" << endl;
    out << r.instructions << endl;
    out << "-----" << endl;

    out << "Likes: " << r.likes << endl;
    out << "-----" << endl;
    out << "Average Rating: " << r.ratings << endl;
    out << "-----" << endl;
    out << "Added By: " << r.addedBy << endl;
    out << "-----" << endl;

    return out;
}
```

(a) << Operator

```
ostream &operator<<(ostream &out, User &u)
{
    out << u.username << " " << u.password;
    return out;
}
```

(b) << Operator

```
istream &operator>>(istream &in, User &u)
{
    in >> u.username >> u.password;
    return in;
}
```

(c) >> Operator

```
bool operator==(const Recipe &r1, const Recipe &r2) // Print the common ingredient
{
    cout << "The common Ingredients in the given two recipes are:" << endl;
    bool checkCommon = false;

    for (int i = 0; i < r1.ingredients.size(); ++i)
    {
        for (int j = 0; j < r2.ingredients.size(); ++j)
        {
            if (r1.ingredients[i].first == r2.ingredients[j].first)
            {
                cout << r1.ingredients[i].first << endl;
                checkCommon = true;
            }
        }
    }
    cout << endl;
    return checkCommon;
}
```

(d) == Operator

Figure 7: Operator Overloading



9 Implemented Features in App

In the following page, we describe two functions that allow users to modify the ingredients of a recipe: [addIngredients](#) and [removeIngredients](#). These functions allow the user to add or remove ingredients in the recipes they have created.

addIngredients Function

```
void Recipe::addIngredients(const string &takeIngredient, const string &quantity)
{
    ingredients.push_back(make_pair(takeIngredient, quantity));
}
```

Figure 8: addIngredients Function

removeIngredients Function

```
void Recipe::removeIngredients(const string &removeIngredients)
{
    for (int i = 0; i < ingredients.size(); i++)
    {
        if (ingredients[i].first == removeIngredients)
        {
            ingredients.erase(ingredients.begin() + i);
            break;
        }
    }
}
```

Figure 9: removeIngredients Function

Conditions

Both functions **only apply to recipes created by the currently logged-in user**. This ensures that users can only modify the ingredients of their own recipes and prevents unauthorized changes to other users' recipes.

10 Implemented Features Contd..

In the following page, we describe functions like `increaseLikes` and `addRatings`.

IncreaseLikes

The `increaseLikes()` method adds **1 to the total likes**, indicating the recipe's popularity.

```
void Recipe::increaseLikes()
{
    likes++;
}
```

Figure 10: IncreaseLikes function

addRatings

The `addRatings(double addedRating)` method manages ratings by:

- **Counting Ratings:** Each new rating increases the `ratingCount` by 1.
- **Updating Average Rating:** Uses this formula to recalculate the average each time:

$$\text{ratings} = \frac{(\text{ratings} \times \text{ratingCount} + \text{addedRating})}{(\text{ratingCount} + 1)}$$

```
void Recipe::addRatings(double addedRating)
{
    ratingCount++;
    ratings = (ratings * ratingCount + addedRating) / (ratingCount + 1);
}
```

Figure 11: addRatings Function

11 Implemented Features Contd..

11.1 AddRecipe

1. Create Recipe:

- A new Recipe object is created using user input (name, ingredients, instructions, etc.).

2. Open File:

- Opens a file named addedRecipe.txt in **append mode** using ofstream.
- If the file fails to open, it shows an **error message** and exits.

3. Check for Duplicates:

- Uses the function isRecipeAdded() to check if a recipe with the same name already exists.
- If a duplicate is found, it informs the user and **skips saving** the recipe.

4. Save Recipe:

- **To File:** Adds the new recipe to addedRecipe.txt.
- **To Vector:** Adds the recipe to a vector called recipes for **fast access**.

5. Close File:

- Ensures the file is properly closed after writing to prevent data corruption.

```
Recipe r1(name, ingredients, instruct, 0, 0, u, h, m, s);

ofstream out("addedRecipe.txt", ios::app);
if (!out)
{
    cerr << "Error opening file for writing." << endl;
    return;
}
if (isRecipeAdded(r1.getRecipeName()))
{
    cout << "This Recipe already exist.please have a look at all recipes" << endl;
    continue;
}

out << r1;
recipes.push_back(r1);
out.close();
```

Figure 12: addRecipe function

12 Implemented Features Contd..

12.1 isRecipeAdded

- The function `isRecipeAdded()` is used to check if a recipe with the same name already exists.
- If a duplicate is found, it informs the user and **skips saving** the recipe.

```
Recipe r1(name, ingredients, instruct, 0, 0, u, h, m, s);

ofstream out("addedRecipe.txt", ios::app);
if (!out)
{
    cerr << "Error opening file for writing." << endl;
    return;
}
if (isRecipeAdded(r1.getRecipeName()))
{
    cout << "This Recipe already exist, please have a look at all recipes" << endl;
    continue;
}

out << r1;
recipes.push_back(r1);
out.close();
```

Figure 13: isRecipeAdded Function

12.2 Recipe of the Day

- The program checks if the recipe list is **empty**; if not, it seeds the **random number generator**, selects a random recipe, and displays it as the **"Recipe of the Day"**.

```
case 6: // "Recipe of the Day"
{
    if (recipes.empty())
    {
        cout << "No recipes available!" << endl;
        break;
    }

    srand(time(0)); // Seed the random number generator
    int randomIndex = rand() % recipes.size(); // Generate a random index

    cout << "Recipe of the Day: " << endl;
    cout << recipes[randomIndex] << endl; // Display the randomly selected recipe
    break;
}
```

Figure 14: Recipe of the Day

13 Test drive(Outputs)

```
Enter 1 to Signup into our app
Enter 2 to login
Enter 3 to exit
□
```

Figure 15: Switch Cases

```
Enter 1 to add your Recipe
Enter 2 to display all recipe
Enter 3 to search and display particular recipe
Enter 4 to display common ingredients between two recipe
Enter 5 to sort/filter recipes based on likes/ratings
Enter 6 to get recipe of the day
Enter 7 to break
```

Figure 16: Second Switch cases

```
Enter 1 to give a rating to this recipe.
Enter 2 to like this recipe.
Enter 3 to add more ingredients to this recipe
Enter 4 to remove ingredients to this recipe
Enter 5 to exit to the main menu.
□
```

Figure 17: Third Switch cases

```
Enter the name of the recipe to be displayed:
Pancakes
=====
                Recipe Details
=====
🍳 Recipe Name: Pancakes
-----
🥄 Ingredients:
-----
Name      |      Quantity
-----
Flour           2 cups
Milk            1 cup
-----
📖 Preparation Time:
-----
0 hours, 15 minutes, 0 seconds
-----
📖 Cooking Instructions:
-----
Mix ingredients and cook.
-----
👍 Likes: 5
-----
★ Average Rating: 4
-----
👤 Added By: rohit
=====
```

Figure 18: Display a Recipe

```
Enter the name of the 1st recipe: Pancakes
Enter the name of the 2nd recipe: Overnight Oats
The common Ingredients in the given two recipes are:
Milk
```

Figure 19: Common Ingredients