Collaborative To-Do list

Session 2023 - 2027

Submitted by:

Muhammad Omer 2023-CS-68

Supervised by:

Muhammad Awais

Course:

CSC-102 Programming Fundamentals

Department of Computer Science

University of Engineering and Technology

Lahore Pakistan

Here you can find the major parts of your Proposal documentation

- ### Short Description of your project

  I wanted to create a collaborative application that boosts the productivity of the people using it.So I created this to do list which is both minimal and collaborative.In it users can not only add tasks to their to do list but also to their peers to do list and request help on tasks they cannot finish alone.

- ### Users of Application

  I wanted this applications to be used by peers so there is no fixed hierarchy. All of the users are on the same level of authority and do not have the capability to change or see any of the other users data. All users have their own files in which data is their data is stored.

- ### Functional Requirements

  The user can do the following functionalities with this application.

  1. Add tasks to himself
  2. View his To-Do list
  3. Remove his tasks
  4. Mark his completed tasks done
  5. View other the list of other peers using this app
  6. Request other peers to help on task
  7. See and mark completed other peoples requested tasks

- ### Wireframes

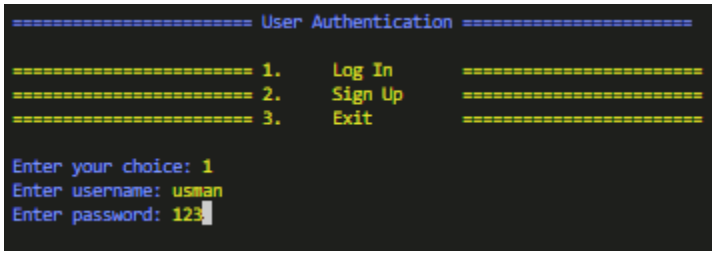


Figure 1: Login Screen

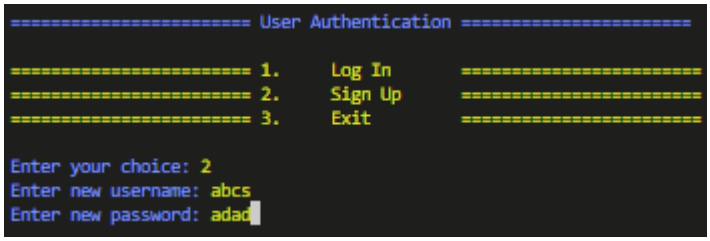


Figure 2: sign up menu

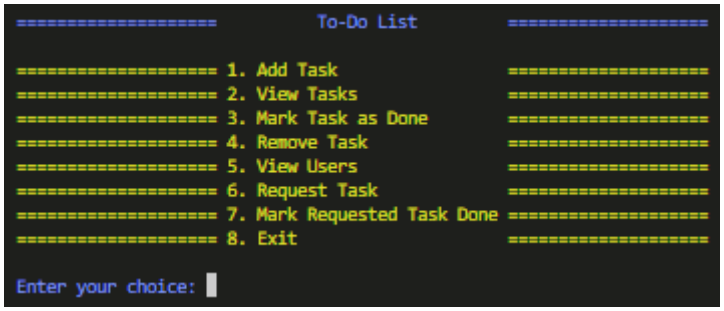


Figure 3: User Main Menu Screen

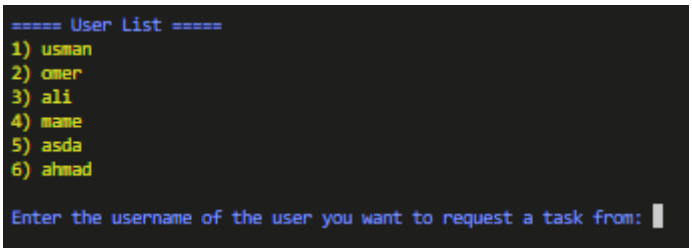


Figure 4: Requesting task menu

- ## Data Structures (Parallel Arrays)

```
string usernames[MAX_USERS];

string passwords[MAX_USERS];

string requestList[MAX_REQUESTS];

int userCount = 0;

int requestCount = 0;
```

- ## Function Prototypes

Void addTask(const string& username, const string& task, bool taskDone[], string taskList[], int& taskCount);

void viewTasks(const string& username, const bool taskDone[], const string taskList[], int taskCount);

void removeTask(const string& username, string taskList[], bool taskDone[], int& taskCount);

void markTaskDone(const string& username, string taskList[], bool taskDone[], int taskCount);

void loadTasksFromFile(const string& username, string taskList[], bool taskDone[], int& taskCount);

void saveTasksToFile(const string& username, const string taskList[], const bool taskDone[], int taskCount);

void requestTask(const string& requester, const string usernames[], string requestList[], int& requestCount);

void viewRequestedTasks(const string& username, const string requestList[], int requestCount);

void markRequestedTaskDone(const string& currentUser, string requestList[], int& requestCount);

void loadRequestsFromFile(const string& username, string requestList[], int& requestCount);

void saveRequestsToFile(const string& username, const string requestList[], int requestCount);

void addUser(string usernames[], string passwords[], int& userCount, const string& username, const string& password);

bool authenticateUser(const string usernames[], const string passwords[], int userCount, string& currentUser);

void saveUsersToFile(const string usernames[], const string passwords[], int userCount);

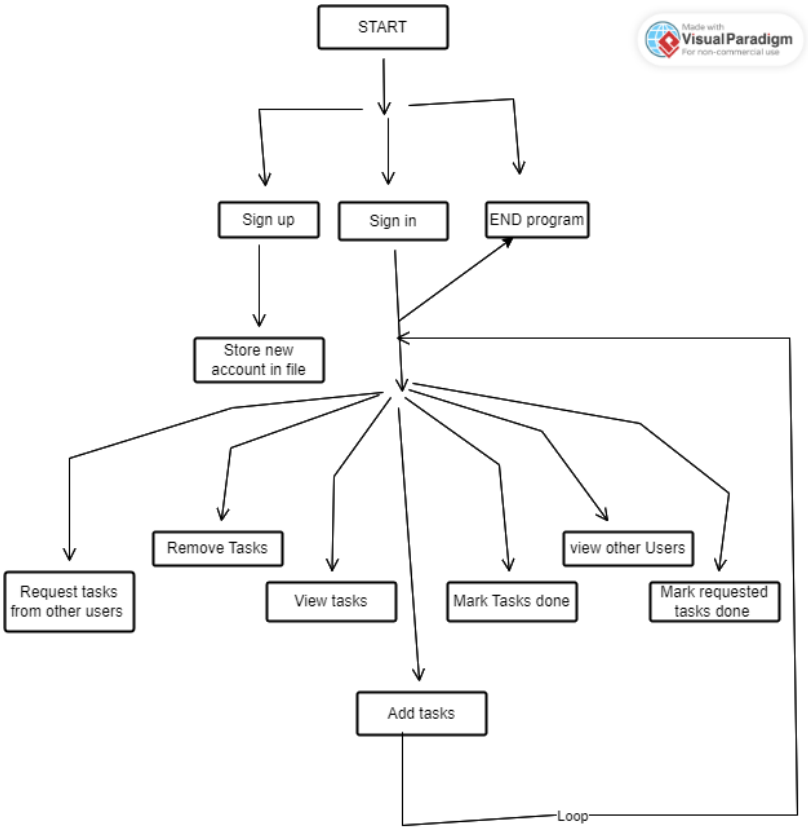
void loadUsersFromFile(string usernames[], string passwords[], int& userCount);

void viewUsers(const string usernames[], int userCount);

int ascii_to_int(string convert_to_int);

void resizeConsole(int width, int height);

- **Functions Working Flow**



CODE:

```
#include <iostream>
#include <fstream>
#include <string>
#include <conio.h>
#include <windows.h>

using namespace std;
//constants
const int MAX_TASKS = 20;
const int MAX_REQUESTS = 20;
const int MAX_USERS = 10;
const int MAX_USERNAME_LENGTH = 50;
const int MAX_PASSWORD_LENGTH = 50;

HANDLE color = GetStdHandle(STD_OUTPUT_HANDLE);

void addTask(const string& username, const string& task, bool taskDone[], string taskList[], int& taskCount);
void viewTasks(const string& username, const bool taskDone[], const string taskList[], int taskCount);
void removeTask(const string& username, string taskList[], bool taskDone[], int& taskCount);
void markTaskDone(const string& username, string taskList[], bool taskDone[], int taskCount);
void loadTasksFromFile(const string& username, string taskList[], bool taskDone[], int& taskCount);
void saveTasksToFile(const string& username, const string taskList[], const bool taskDone[], int taskCount);
void requestTask(const string& requester, const string usernames[], string requestList[], int& requestCount);
void viewRequestedTasks(const string& username, const string requestList[], int requestCount);
void markRequestedTaskDone(const string& currentUser, string requestList[], int& requestCount);
void loadRequestsFromFile(const string& username, string requestList[], int& requestCount);
void saveRequestsToFile(const string& username, const string requestList[], int requestCount);
void addUser(string usernames[], string passwords[], int& userCount, const string& username, const string& password);
bool authenticateUser(const string usernames[], const string passwords[], int userCount, string& currentUser);
void saveUsersToFile(const string usernames[], const string passwords[], int userCount);
void loadUsersFromFile(string usernames[], string passwords[], int& userCount);
```

```
void viewUsers(const string usernames[], int userCount);
int ascii_to_int(string convert_to_int);

int main() {

   string usernames[MAX_USERS];          //used to store usernames
   string passwords[MAX_USERS];          //used to store passwords
   string requestList[MAX_REQUESTS];     //used to store requests
   int userCount = 0;                    //used to store user count
   int requestCount = 0;                 //used to store requst numbers

   loadUsersFromFile(usernames, passwords, userCount);          //load userlist form file
   string temp_choice;                                  //used to store the below choice temporarily
   int choice;                                         //is used for the case change
   string currentUser;                                 //is used to store the current user

   do {                                               //Do while loop is better to use for such cases
      system("cls");                                  //Code for the login menu
      SetConsoleTextAttribute(color, 9);
      cout << "\n======================== User Authentication ========================" << endl<<endl;
      SetConsoleTextAttribute(color, 14);
      cout <<  "======================== 1.    Log In      ========================" << endl;
      cout <<  "======================== 2.    Sign Up     ========================" << endl;
      cout <<  "======================== 3.    Exit        ========================" << endl;
      SetConsoleTextAttribute(color, 9);
      cout <<endl<< "Enter your choice: ";
      SetConsoleTextAttribute(color, 14);
      cin >> temp_choice;
      choice=ascii_to_int(temp_choice);
      switch (choice) {
         case 1:
            if (authenticateUser(usernames, passwords, userCount, currentUser)) {   //this function authenticates user and checks
if he has entered the name creds
```

```
cout << "Welcome, " << currentUser << "!" << endl;

string taskList[MAX_TASKS];
bool taskDone[MAX_TASKS];
int taskCount = 0;

loadTasksFromFile(currentUser, taskList, taskDone, taskCount);

string requestedTasks[MAX_TASKS];
int requestedTaskCount = 0;
string temp_taskchoice;
int taskChoice;

do {                                        //second do while loop for user specific menu
   system("cls");
   SetConsoleTextAttribute(color, 9);
   cout << "\n====================        To-Do List       ====================" << endl<<endl;
   SetConsoleTextAttribute(color, 14);
   cout <<   "==================== 1. Add Task              ====================" << endl;
   cout <<   "==================== 2. View Tasks            ====================" << endl;
   cout <<   "==================== 3. Mark Task as Done      ====================" << endl;
   cout <<   "==================== 4. Remove Task           ====================" << endl;
   cout <<   "==================== 5. View Users            ====================" << endl;
   cout <<   "==================== 6. Request Task          ====================" << endl;
   cout <<   "==================== 7. Mark Requested Task Done ====================" << endl;
   cout <<   "==================== 8. Exit                  ====================" << endl;
   SetConsoleTextAttribute(color, 9);
   cout <<endl<< "Enter your choice: ";
   SetConsoleTextAttribute(color, 14);
   cin >> temp_taskchoice;
   taskChoice = ascii_to_int(temp_taskchoice);
   switch (taskChoice) {                            //switch statement to redirect to the required function
```

```
        case 1: {
            system("cls");
            string newTask;
            SetConsoleTextAttribute(color, 9);
            cout << "Enter new task: ";
            SetConsoleTextAttribute(color, 14);
            cin.ignore();
            getline(cin, newTask);
            addTask(currentUser, newTask, taskDone, taskList, taskCount);
            cout << endl;
            SetConsoleTextAttribute(color, 9);
            cout << "Press any key to continue";
            getch();
            break;
        }
        case 2:
            system("cls");
            viewTasks(currentUser, taskDone, taskList, taskCount);
            cout << endl;
            SetConsoleTextAttribute(color, 14);
            cout << "Press any key to continue";
            getch();
            break;
        case 3:
            system("cls");
            viewTasks(currentUser, taskDone, taskList, taskCount);
            cout << endl;
            markTaskDone(currentUser, taskList, taskDone, taskCount);
            cout << endl;
            SetConsoleTextAttribute(color, 14);
            cout << "Press any key to continue";
            getch();
```

```
            break;
        case 4:
            system("cls");
            viewTasks(currentUser, taskDone, taskList, taskCount);
            cout << endl;
            removeTask(currentUser, taskList, taskDone, taskCount);
            cout << endl;
            cout << "Press any key to continue";
            getch();
            break;
        case 5:
            system("cls");
            viewUsers(usernames, userCount);
            cout << endl;
            cout << "Press any key to continue";
            getch();
            break;
        case 6:
            system("cls");
            viewUsers(usernames, userCount);
            cout << endl;
            requestTask(currentUser, usernames, requestList, requestCount);
            cout << endl;
            cout << "Press any key to continue";
            getch();
            break;
        case 7:
            system("cls");
            viewRequestedTasks(currentUser, requestList, requestCount);
            cout << endl;
            markRequestedTaskDone(currentUser, requestList, requestCount);
```

```
            cout << endl;
            cout << "Press any key to continue";
            getch();
            break;

        case 8:
            system("cls");
            cout << "Goodbye!" << endl;
            getch();
            break;

        default:
            system("cls");
            cout << endl;
            SetConsoleTextAttribute(color, 12);
            cout << "Invalid choice. Please try again." << endl;
            cout << endl;
            SetConsoleTextAttribute(color, 14);
            cout << "Press any key to continue";
            getch();
      }

    } while (taskChoice != 8);

    break;
  } else {
    SetConsoleTextAttribute(color, 12);
    cout << "Authentication failed. Please try again." << endl;
    getch();
  }
  break;
case 2: {                       //case for sign up
```

```
            SetConsoleTextAttribute(color, 9);
            string newUsername, newPassword;
            cout << "Enter new username: ";
            SetConsoleTextAttribute(color, 14);
            cin >> newUsername;
            SetConsoleTextAttribute(color, 9);
            cout << "Enter new password: ";
            SetConsoleTextAttribute(color, 14);
            cin >> newPassword;

            addUser(usernames, passwords, userCount, newUsername, newPassword);    //used to add user
            saveUsersToFile(usernames, passwords, userCount);                  //used to save the added user details to file
            break;
        }
        case 3:                                          //exit program case
        SetConsoleTextAttribute(color, 14);
            cout << "Goodbye!" << endl;
            break;
        default:                                          //validation
            SetConsoleTextAttribute(color, 12);
            cout << "Invalid choice. Please try again." << endl;
            getch();
      }

   } while (choice != 3);

   return 0;
}

void addTask(const string& username, const string& task, bool taskDone[], string taskList[], int& taskCount) {
   ofstream outputFile(username + "_tasks.txt", ios::app);
```

```
    if (outputFile.is_open()) {
        // Add the new task to the task list
        taskList[taskCount] = task;
        // Mark the new task as not done by default
        taskDone[taskCount] = false;

        // Append the task to the file
        outputFile << taskList[taskCount] << " | " << taskDone[taskCount] << endl;

        // Increment the task count
        taskCount++;

        outputFile.close();
    } else {
        cout << "Error opening file for tasks." << endl;
    }
}
void viewTasks(const string& username, const bool taskDone[], const string taskList[], int taskCount) {
    ifstream inputFile(username + "_tasks.txt");

    if (inputFile.is_open()) {
        SetConsoleTextAttribute(color, 9);
        cout << "\n===== Task List for " << username << " =====" << endl<<endl;
        for (int i = 0; i < taskCount; ++i) {
            SetConsoleTextAttribute(color, 14);
            cout << i + 1 << ") " << taskList[i] << " - " << (taskDone[i] ? "Done" : "Not Done") << endl;
        }

        inputFile.close();
    } else {
        SetConsoleTextAttribute(color, 12);
        cout << "Error opening file for tasks." << endl;
```

```
    }
}
void removeTask(const string& username, string taskList[], bool taskDone[], int& taskCount) {
    int taskNumber;
    SetConsoleTextAttribute(color, 9);
    cout << "Enter the task number to remove: ";
    SetConsoleTextAttribute(color, 14);
    cin >> taskNumber;

    if (taskNumber > 0 && taskNumber <= taskCount) {
        // Shift elements to remove the task
        for (int i = taskNumber - 1; i < taskCount - 1; ++i) {
            taskList[i] = taskList[i + 1];
            taskDone[i] = taskDone[i + 1];
        }

        // Decrement the task count
        taskCount--;

        // Update the file after removing the task
        saveTasksToFile(username, taskList, taskDone, taskCount);
        SetConsoleTextAttribute(color, 10);
        cout << "Task removed." << endl;
    } else {
        SetConsoleTextAttribute(color, 12);
        cout << "Invalid task number." << endl;
    }
}
void markTaskDone(const string& username, string taskList[], bool taskDone[], int taskCount) {
    int taskNumber;
    SetConsoleTextAttribute(color, 9);
    cout << "Enter the task number to mark as done: ";
```

```
    SetConsoleTextAttribute(color, 14);
    cin >> taskNumber;

    if (taskNumber > 0 && taskNumber <= taskCount) {
        taskDone[taskNumber - 1] = true;

        // Update the file after marking the task as done
        saveTasksToFile(username, taskList, taskDone, taskCount);
        SetConsoleTextAttribute(color, 10);
        cout << "Task marked as done." << endl;
    } else {
        SetConsoleTextAttribute(color, 12);
        cout << "Invalid task number." << endl;
    }
}
void loadTasksFromFile(const string& username, string taskList[], bool taskDone[], int& taskCount) {
    ifstream inputFile(username + "_tasks.txt");

    if (inputFile.is_open()) {
        taskCount = 0;
        while (getline(inputFile, taskList[taskCount], '|')) {
            inputFile >> taskDone[taskCount];
            inputFile.ignore(); // Consume the newline character
            taskCount++;
        }

        inputFile.close();
    } else {
        cout << "Error opening file for tasks." << endl;
    }
}
void saveTasksToFile(const string& username, const string taskList[], const bool taskDone[], int taskCount) {
```

```
    ofstream outputFile(username + "_tasks.txt");

    if (outputFile.is_open()) {
        for (int i = 0; i < taskCount; ++i) {
            outputFile << taskList[i] << " | " << taskDone[i] << endl;
        }

        outputFile.close();
    } else {
        cout << "Error opening file for tasks." << endl;
    }
}
void markRequestedTaskDone(const string& currentUser, string requestList[], int& requestCount) {
    int requestNumber;
    SetConsoleTextAttribute(color, 9);
    cout << "Enter the task number to mark as done: ";
    SetConsoleTextAttribute(color, 14);
    cin >> requestNumber;

    if (requestNumber > 0 && requestNumber <= requestCount) {
        // Remove the marked requested task
        for (int i = requestNumber - 1; i < requestCount - 1; ++i) {
            requestList[i] = requestList[i + 1];
        }

        // Decrement the request count
        requestCount--;

        // Save the modified request list to the file
        saveRequestsToFile(currentUser, requestList, requestCount);
        SetConsoleTextAttribute(color, 10);
        cout << "Requested task marked as done." << endl;
```

```
    } else {
        SetConsoleTextAttribute(color, 12);
        cout << "Invalid task number." << endl;
    }
}
void requestTask(const string& requester, const string usernames[], string requestList[], int& requestCount) {
    string assignee;
    SetConsoleTextAttribute(color, 9);
    cout << "Enter the username of the user you want to request a task from: ";
    SetConsoleTextAttribute(color, 14);
    cin >> assignee;

    bool userFound = false;
    for (int i = 0; i < MAX_USERS; ++i) {
        if (usernames[i] == assignee) {
            userFound = true;
            string requestedTask;
            SetConsoleTextAttribute(color, 9);
            cout << "Enter the task you want to request: ";
            cin.ignore();
            SetConsoleTextAttribute(color, 14);
            getline(cin, requestedTask);

            // Add the requested task to the assignee's request list
            requestList[requestCount] = assignee + " requested: " + requestedTask;
            requestCount++;
            SetConsoleTextAttribute(color, 14);
            cout << "Task request sent to " << assignee << "." << endl;
            break;
        }
    }
```

```
    if (!userFound) {
        SetConsoleTextAttribute(color, 12);
        cout << "User not found. Please enter a valid username." << endl;
    }
}
void viewRequestedTasks(const string& username, const string requestList[], int requestCount) {
    SetConsoleTextAttribute(color, 9);
    cout << "\n===== Requested Tasks =====" << endl;
    SetConsoleTextAttribute(color, 14);
    for (int i = 0; i < requestCount; ++i) {
        if (requestList[i].find(username + " requested:") != string::npos) {
            cout << i + 1 << ") " << requestList[i] << endl;
        }
    }
}
void loadRequestsFromFile(const string& username, string requestList[], int& requestCount) {
    ifstream inputFile(username + "_requests.txt");

    if (inputFile.is_open()) {
        requestCount = 0;
        while (getline(inputFile, requestList[requestCount])) {
            requestCount++;
        }

        inputFile.close();
    } else {
        cout << "Error opening file for requests." << endl;
    }
}
void saveRequestsToFile(const string& username, const string requestList[], int requestCount) {
    ofstream outputFile(username + "_requests.txt");
```

```
    if (outputFile.is_open()) {
        for (int i = 0; i < requestCount; ++i) {
            outputFile << requestList[i] << endl;
        }

        outputFile.close();
    } else {
        cout << "Error opening file for requests." << endl;
    }
}
void addUser(string usernames[], string passwords[], int& userCount, const string& username, const string& password) {
    if (userCount < MAX_USERS) {
        usernames[userCount] = username;
        passwords[userCount] = password;
        userCount++;
        cout << "User added: " << username << endl;
    } else {
        cout << "Maximum number of users reached." << endl;
    }
}
bool authenticateUser(const string usernames[], const string passwords[], int userCount, string& currentUser) {
    string enteredUsername, enteredPassword;
    SetConsoleTextAttribute(color, 9);
    cout << "Enter username: ";
    SetConsoleTextAttribute(color, 14);
    cin >> enteredUsername;
    SetConsoleTextAttribute(color, 9);
    cout << "Enter password: ";
    SetConsoleTextAttribute(color, 14);
    cin >> enteredPassword;

    for (int i = 0; i < userCount; ++i) {
```

```
        if (usernames[i] == enteredUsername && passwords[i] == enteredPassword) {
            currentUser = enteredUsername;
            return true;
        }
    }

    return false;
}
void saveUsersToFile(const string usernames[], const string passwords[], int userCount) {
    ofstream outputFile("users.txt");

    if (outputFile.is_open()) {
        for (int i = 0; i < userCount; ++i) {
            outputFile << usernames[i] << " " << passwords[i] << endl;
        }

        outputFile.close();
    } else {
        cout << "Error opening file for users." << endl;
    }
}
void loadUsersFromFile(string usernames[], string passwords[], int& userCount) {
    ifstream inputFile("users.txt");

    if (inputFile.is_open()) {
        while (inputFile >> usernames[userCount] >> passwords[userCount]) {
            userCount++;
        }

        inputFile.close();
    } else {
        cout << "Error opening file for users." << endl;
```

```
    }
}
int ascii_to_int(string convert_to_int)
{
    int result = 0;

    for (char ch : convert_to_int)
    {
        if (isdigit(ch))
        {
            result = result * 10 + (ch - '0');
        }
    }

    return result;
}
void viewUsers(const string usernames[], int userCount) {
    SetConsoleTextAttribute(color, 9);
    cout << "\n===== User List =====" << endl;
    SetConsoleTextAttribute(color, 14);
    for (int i = 0; i < userCount; ++i) {
        cout << i + 1 << ") " << usernames[i] << endl;
    }
}
```