**University of Central Punjab, Lahore**

**Faculty of Information and Technology**

****

***Group Members***

***L1F22BSCS0381 – Aqsa Mahmood Virk***

***L1F22BSCS0377 – Tehreem Sohail***

***L1F22BSCS0272 – Hafiza Fatima***

***BS Computer Science***

***Subject:***

***DSA***

***Submission To:***

***Saira Shairi***

**Project Report – Semester 4**

***Report on Social Networking Project in C++***

***Abstract:***

This project implements a basic social networking application using C++ as a part of a Data Structures and Algorithms (DSA) course. The application allows users to manage profiles, follow and unfollow other users, and post updates. It uses key data structures, such as binary search trees (BST) for efficient user profile management and linked lists to handle connections between users. Each profile contains essential information such as user ID, name, email, profile picture, and a count of connections. The program includes functionalities for adding, removing, searching, and displaying users, as well as managing following relationships. This report provides an in-depth analysis of the code, its structure, and the algorithms used.

***Introduction:***

The social networking application developed in this project is designed to handle user profiles and their interactions efficiently. Below are the main components and features of the application:

* **UserProfile Class:** Stores information about each user, including user ID, name, email, profile picture, and connection count.
* **LinkedList Class:** Manages the list of connections (followers) for each user using a simple linked list data structure.
* **BinarySearchTree Class:** Handles the storage and retrieval of user profiles using a binary search tree for efficient search operations.
* **File I/O Operations:** Provides functions to save and load user profiles from a file, ensuring data persistence between program executions.
* **User Interface:** Offers a text-based interface for users to add, remove, search, display, follow, unfollow, and post updates.

***Code Guidelines and Features:***

* **User Management:** Users can be added, removed, and searched in the system. Each user profile is stored in a BST to facilitate quick lookups.
* **Connections:** Users can follow or unfollow other users. The connections are managed using a linked list within each user profile node in the BST.
* **Data Persistence:** User profiles and their connections are saved to a file, allowing the application to reload the data on subsequent runs.
* **User Interface:** The program presents a menu-driven interface that guides users through various operations.

***Methodology:***

This section describes the development environment and elaborates on the functionality of each part of the code.

***Development Environment:***

The project was developed using Visual Studio 2022, a comprehensive development environment that supports C++ programming. Visual Studio 2022 was chosen for its robust debugging tools, IntelliSense feature for code completion, and efficient project management capabilities.

***Code Explanation:***

1. **UserProfile Class:**

**Constructors:** Initialize user profiles with default or provided values.

**Attributes:** `userID`, `name`, `email`, `profilePicture`, and `connectionCount`.

1. **LinkedList Class:**

**add():** Adds a new connection to the list.

**Remove():** Removes a specified connection from the list.

**Search():** Searches for a specific connection in the list.

**Display():** Displays all connections in the list.

1. **BinarySearchTree Class:**

**addHelper():** Recursively adds a user profile to the BST.

**removeHelper():** Recursively removes a user profile from the BST.

**searchHelper():** Recursively searches for a user profile by user ID.

**inorderHelper():** Recursively performs an in-order traversal to display user profiles.

**findMin():** Finds the node with the minimum key value (userID) in the BST.

1. **File Operations:**

**saveToFile():** Saves all user profiles to a specified file.

**loadFromFile():** Loads user profiles from a specified file into the BST.

1. **Main Function:**

**User Interface:** Provides a menu for user operations such as adding, removing, searching, displaying users, following and unfollowing users, and posting updates.

**User Input Handling:** Takes user input to perform the selected operation and updates the BST accordingly.

***Conclusion:***

The social networking application developed as part of this project demonstrates the effective use of data structures like binary search trees and linked lists in C++. By leveraging these structures, the application provides efficient management of user profiles and connections. Visual Studio 2019 facilitated the development process with its powerful tools and features. This project serves as a practical application of DSA concepts, reinforcing the importance of choosing appropriate data structures for specific tasks.

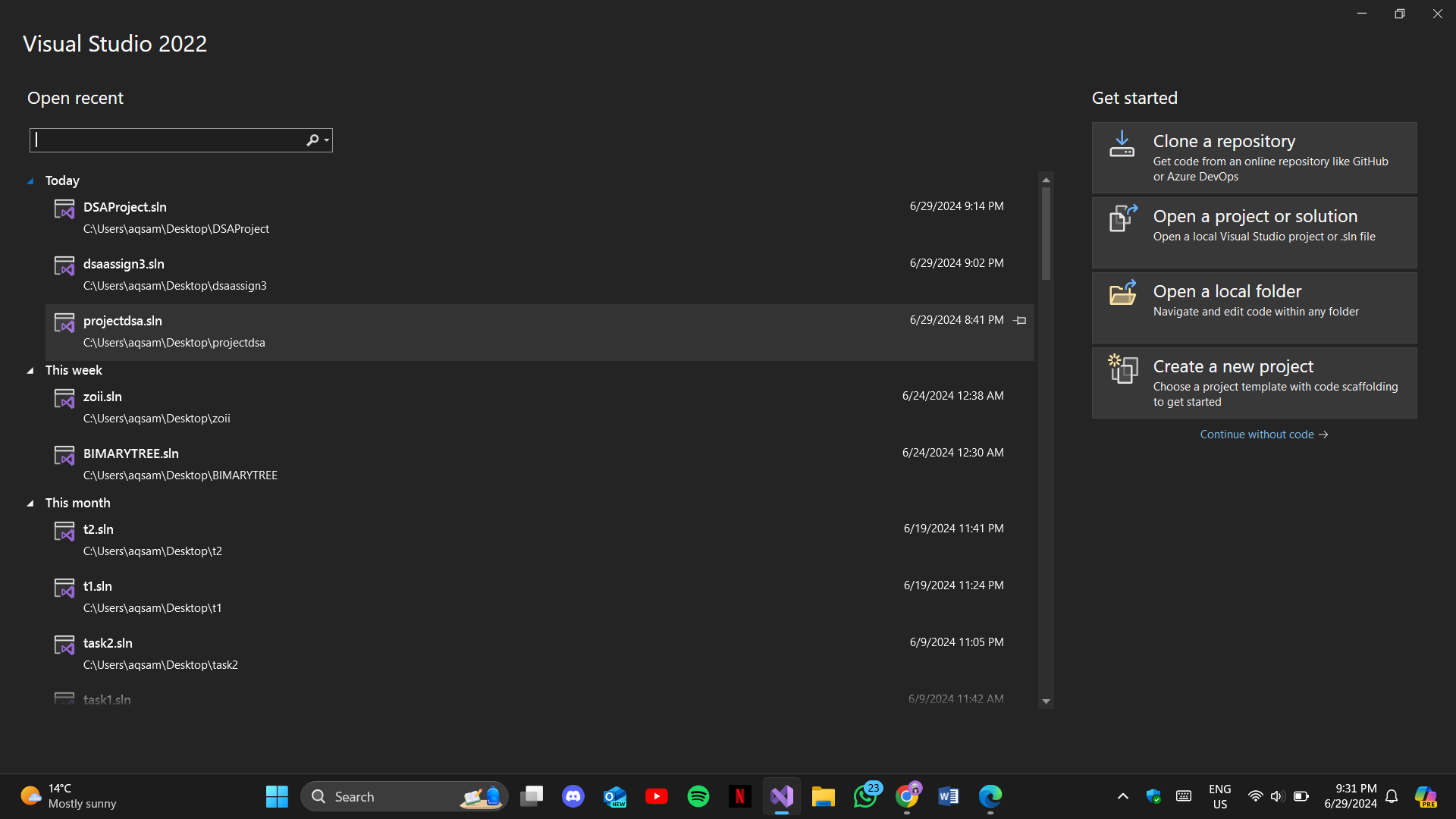
This report outlines the main features and functionalities of the social networking application, providing a comprehensive overview of its design and implementation. The code is structured to ensure efficiency and ease of use, with a focus on key operations such as user management and connections handling.

Now Opening the Visual Studio 2022 to make Social Networking Project in C++.

Few steps are given below:

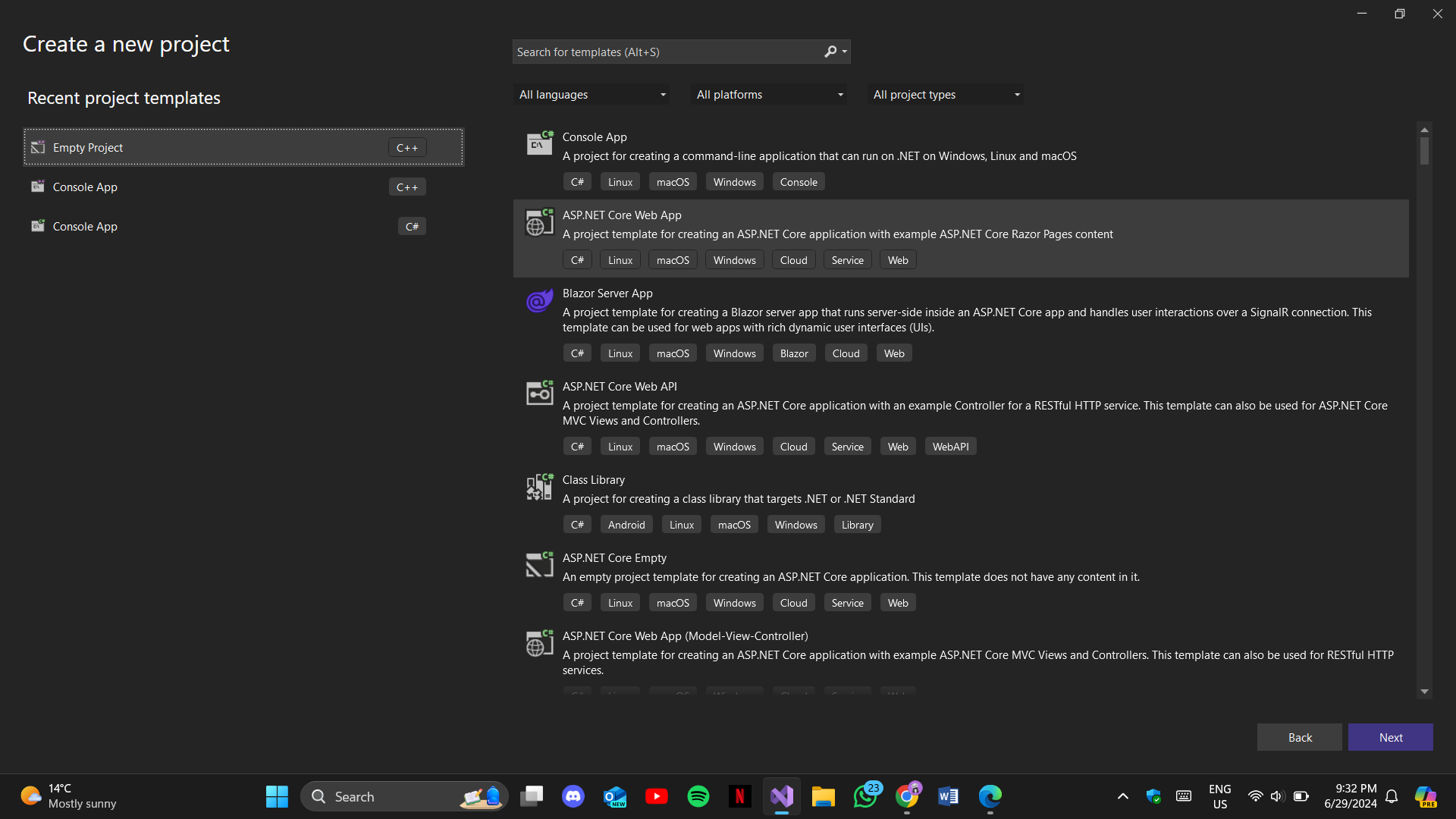
***Step1:***

Select create project to make a project

******

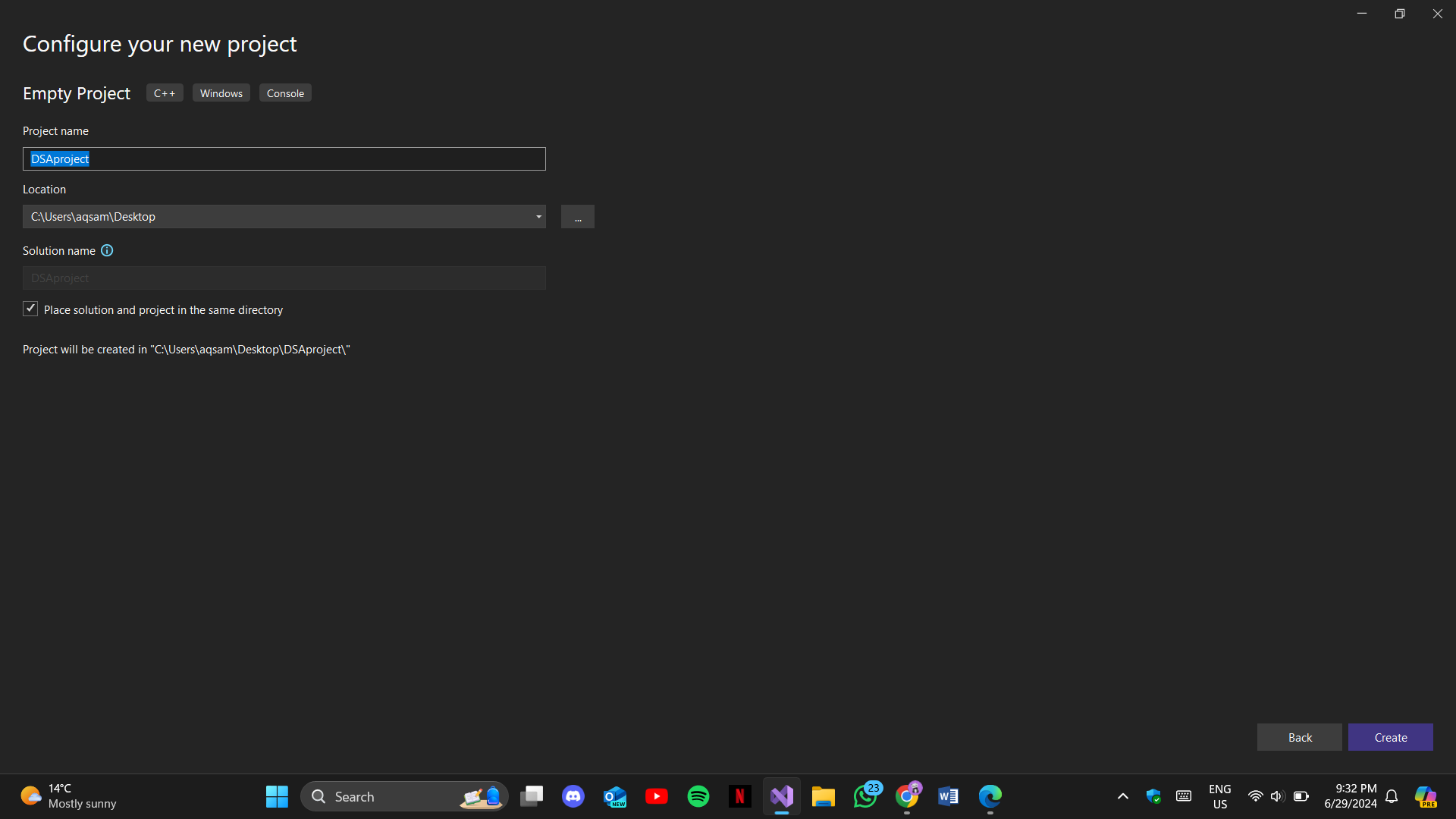
***Step2:***

Select empty project for fresh start



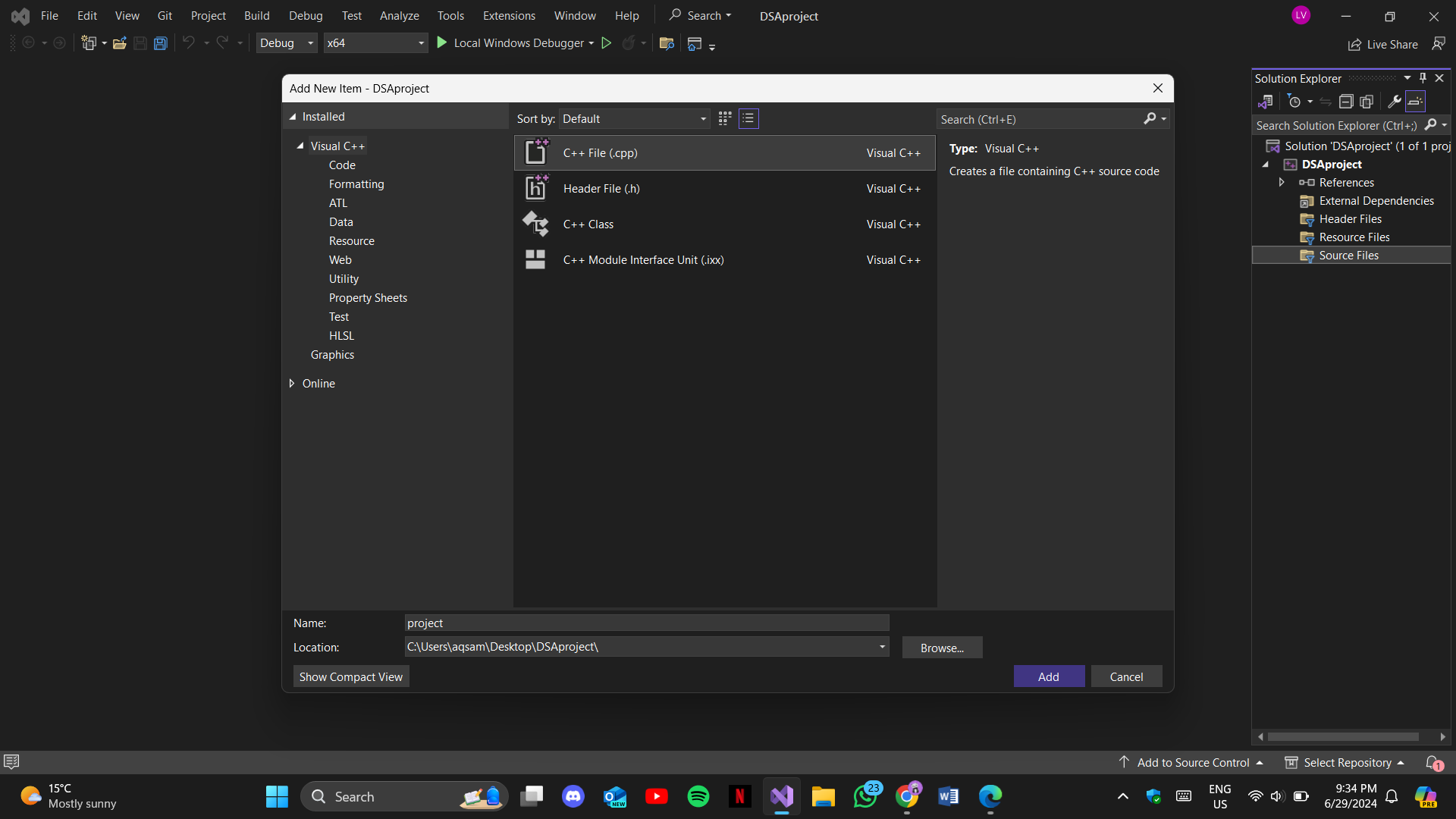
***Step3:***

Name your project



***Step4:***

Select the language for your project

******

***Adding the source code below:***

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

// Define constants

const int MAX\_CONNECTIONS = 100;

const int MAX\_USERS = 1000;

const int MAX\_STRING\_LENGTH = 100;

// UserProfile class to store user information

class UserProfile {

public:

int userID;

string name;

string email;

string profilePicture;

int connections[MAX\_CONNECTIONS];

int connectionCount;

UserProfile() : userID(0), connectionCount(0) {

for (int i = 0; i < MAX\_CONNECTIONS; ++i) {

connections[i] = 0;

}

}

UserProfile(int id, const string& name, const string& email, const string& picture)

: userID(id), name(name), email(email), profilePicture(picture), connectionCount(0) {

for (int i = 0; i < MAX\_CONNECTIONS; ++i) {

connections[i] = 0;

}

}

};

// Node structure for LinkedList

template <typename T>

class Node {

public:

T data;

Node<T>\* next;

Node(const T& data) : data(data), next(nullptr) {}

};

// LinkedList implementation to manage UserProfiles

template <typename T>

class LinkedList {

private:

Node<T>\* head;

public:

LinkedList() : head(nullptr) {}

void add(const T& data) {

Node<T>\* newNode = new Node<T>(data);

if (!head) {

head = newNode;

}

else {

Node<T>\* temp = head;

while (temp->next) {

temp = temp->next;

}

temp->next = newNode;

}

}

bool remove(int userID) {

if (!head) return false;

if (head->data.userID == userID) {

Node<T>\* temp = head;

head = head->next;

delete temp;

return true;

}

Node<T>\* current = head;

while (current->next && current->next->data.userID != userID) {

current = current->next;

}

if (current->next) {

Node<T>\* temp = current->next;

current->next = current->next->next;

delete temp;

return true;

}

return false;

}

Node<T>\* search(int userID) {

Node<T>\* current = head;

while (current) {

if (current->data.userID == userID) {

return current;

}

current = current->next;

}

return nullptr;

}

// Public getter for head

Node<T>\* getHead() const {

return head;

}

};

// Function to save user profiles to a file

void saveToFile(const string& filename, const LinkedList<UserProfile>& userList) {

ofstream outFile(filename);

if (outFile.is\_open()) {

Node<UserProfile>\* current = userList.getHead();

while (current) {

outFile << current->data.userID << " "

<< current->data.name << " "

<< current->data.email << " "

<< current->data.profilePicture << " "

<< current->data.connectionCount << "\n";

current = current->next;

}

outFile.close();

}

}

// Function to load user profiles from a file

void loadFromFile(const string& filename, LinkedList<UserProfile>& userList) {

ifstream inFile(filename);

if (inFile.is\_open()) {

int userID;

string name, email, profilePicture;

int connectionCount;

while (inFile >> userID >> name >> email >> profilePicture >> connectionCount) {

UserProfile user(userID, name, email, profilePicture);

user.connectionCount = connectionCount;

userList.add(user);

}

inFile.close();

}

}

int main() {

LinkedList<UserProfile> userList;

// Load user profiles from file

loadFromFile("users.txt", userList);

while (true) {

cout << "1. Add User\n2. Remove User\n3. Search User\n4. Display Users\n5. Follow User\n6. Unfollow User\n7. Post Update\n8. Exit\n";

int choice;

cin >> choice;

if (choice == 1) {

int userID;

string name, email, picture;

cout << "Enter UserID: ";

cin >> userID;

cout << "Enter name: ";

cin >> name;

cout << "Enter email: ";

cin >> email;

cout << "Enter profile picture: ";

cin >> picture;

userList.add(UserProfile(userID, name, email, picture));

saveToFile("users.txt", userList);

}

else if (choice == 2) {

int userID;

cout << "Enter UserID to remove: ";

cin >> userID;

if (userList.remove(userID)) {

saveToFile("users.txt", userList);

}

else {

cout << "User not found.\n";

}

}

else if (choice == 3) {

int userID;

cout << "Enter UserID to search: ";

cin >> userID;

Node<UserProfile>\* user = userList.search(userID);

if (user) {

cout << "User found: " << user->data.name << " (" << user->data.email << ")\n";

}

else {

cout << "User not found.\n";

}

}

else if (choice == 4) {

Node<UserProfile>\* current = userList.getHead();

while (current) {

cout << "UserID: " << current->data.userID

<< ", Name: " << current->data.name

<< ", Email: " << current->data.email

<< ", Profile Picture: " << current->data.profilePicture

<< ", Connections: " << current->data.connectionCount << "\n";

current = current->next;

}

}

else if (choice == 5) {

int userID, followID;

cout << "Enter your UserID: ";

cin >> userID;

cout << "Enter UserID to follow: ";

cin >> followID;

Node<UserProfile>\* user = userList.search(userID);

if (user && user->data.connectionCount < MAX\_CONNECTIONS) {

user->data.connections[user->data.connectionCount++] = followID;

saveToFile("users.txt", userList);

cout << "Following UserID: " << followID << "\n";

}

else {

cout << "User not found or max connections reached.\n";

}

}

else if (choice == 6) {

int userID, unfollowID;

cout << "Enter your UserID: ";

cin >> userID;

cout << "Enter UserID to unfollow: ";

cin >> unfollowID;

Node<UserProfile>\* user = userList.search(userID);

if (user) {

int\* connections = user->data.connections;

int& count = user->data.connectionCount;

bool found = false;

for (int i = 0; i < count; ++i) {

if (connections[i] == unfollowID) {

for (int j = i; j < count - 1; ++j) {

connections[j] = connections[j + 1];

}

--count;

found = true;

break;

}

}

if (found) {

saveToFile("users.txt", userList);

}

else {

cout << "User not found or not following.\n";

}

}

else {

cout << "User not found.\n";

}

}

else if (choice == 7) {

int userID;

string update;

cout << "Enter your UserID: ";

cin >> userID;

cout << "Enter update: ";

cin.ignore(); // Clear newline character from buffer

getline(cin, update);

// Here, you can store the update in a list or file associated with the user

cout << "Update posted: " << update << "\n";

}

else if (choice == 8) {

break;

}

else {

cout << "Invalid choice. Please try again.\n";

}

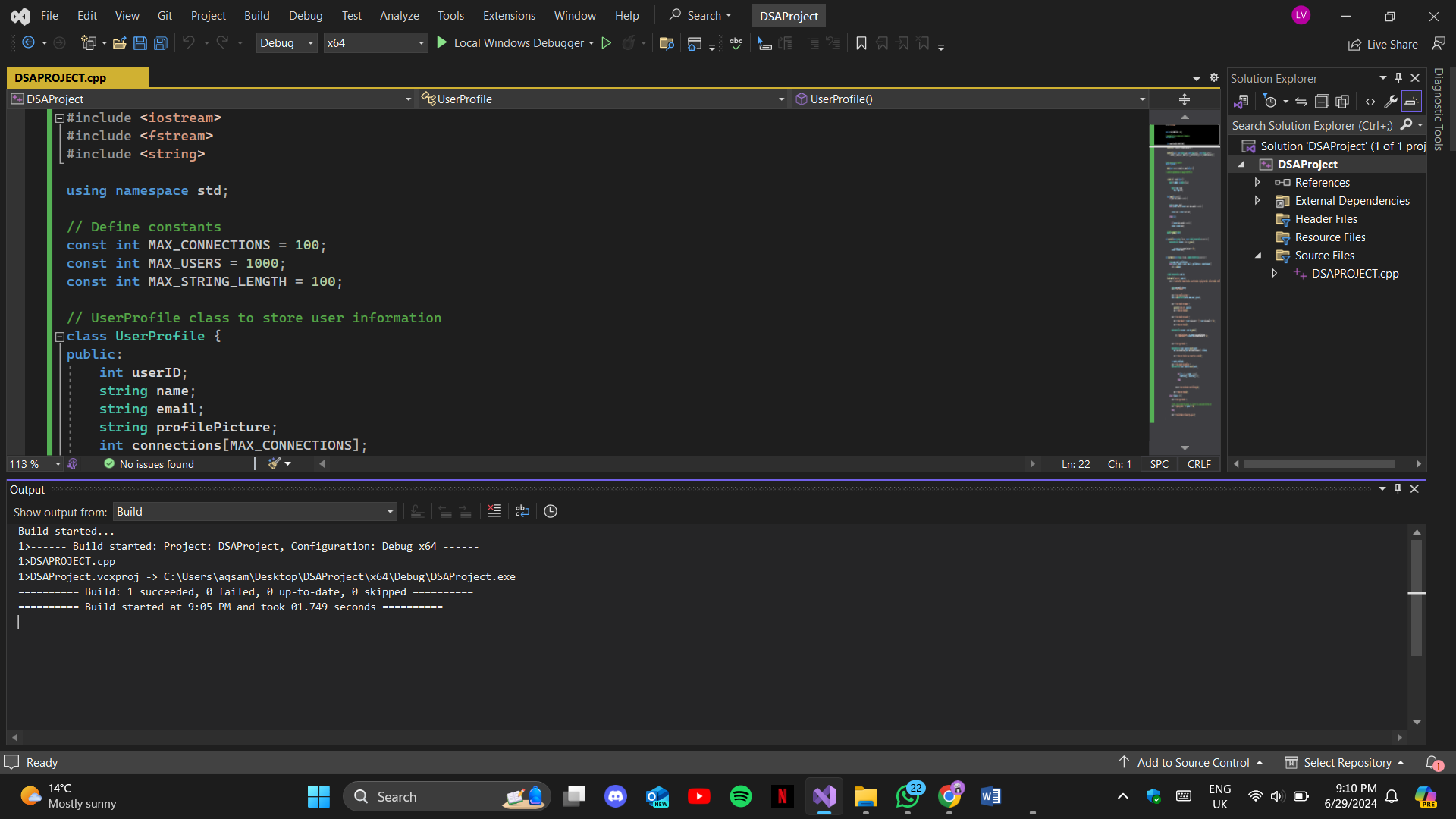
}

return 0;

}

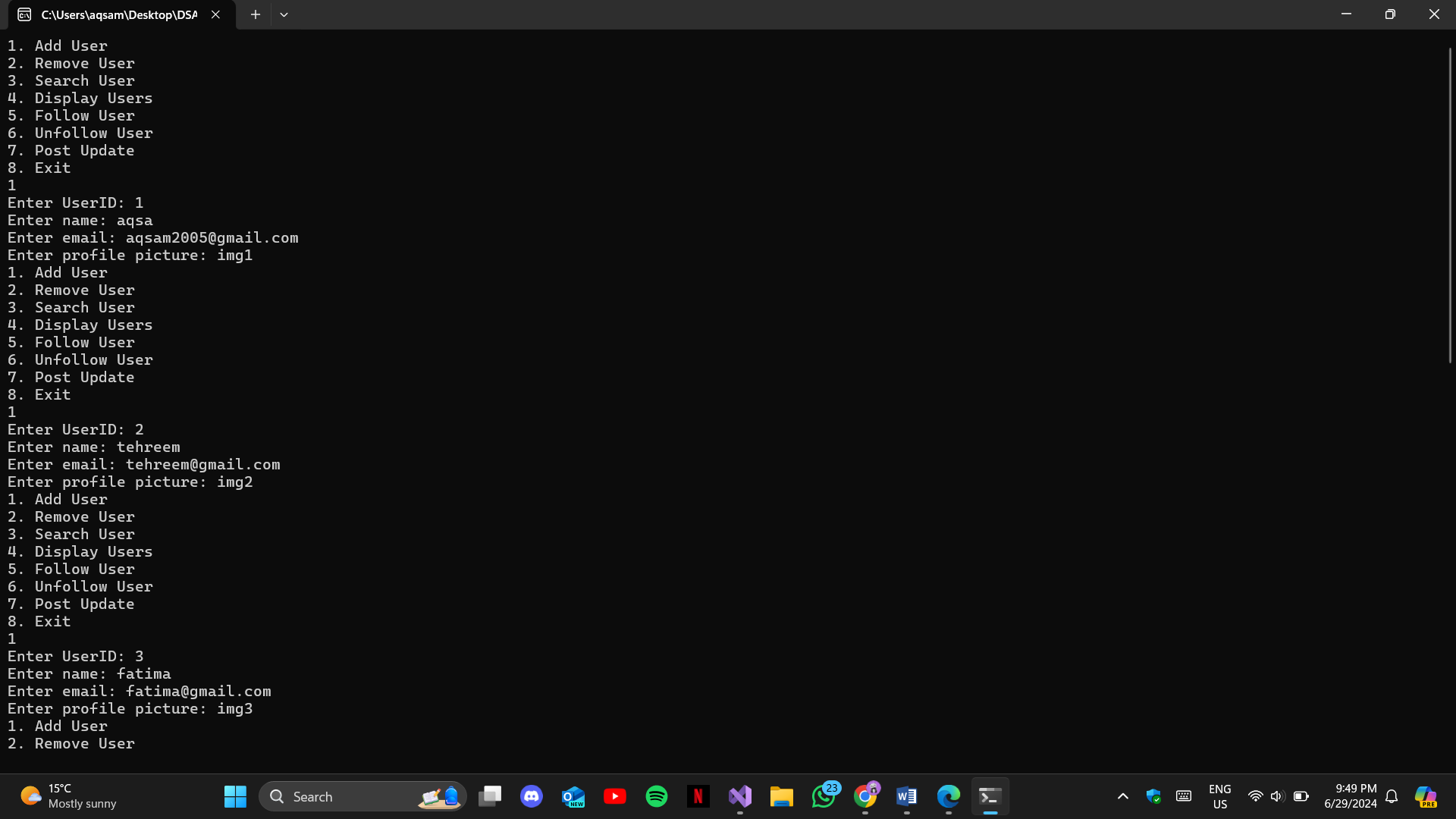
***Successful build:***

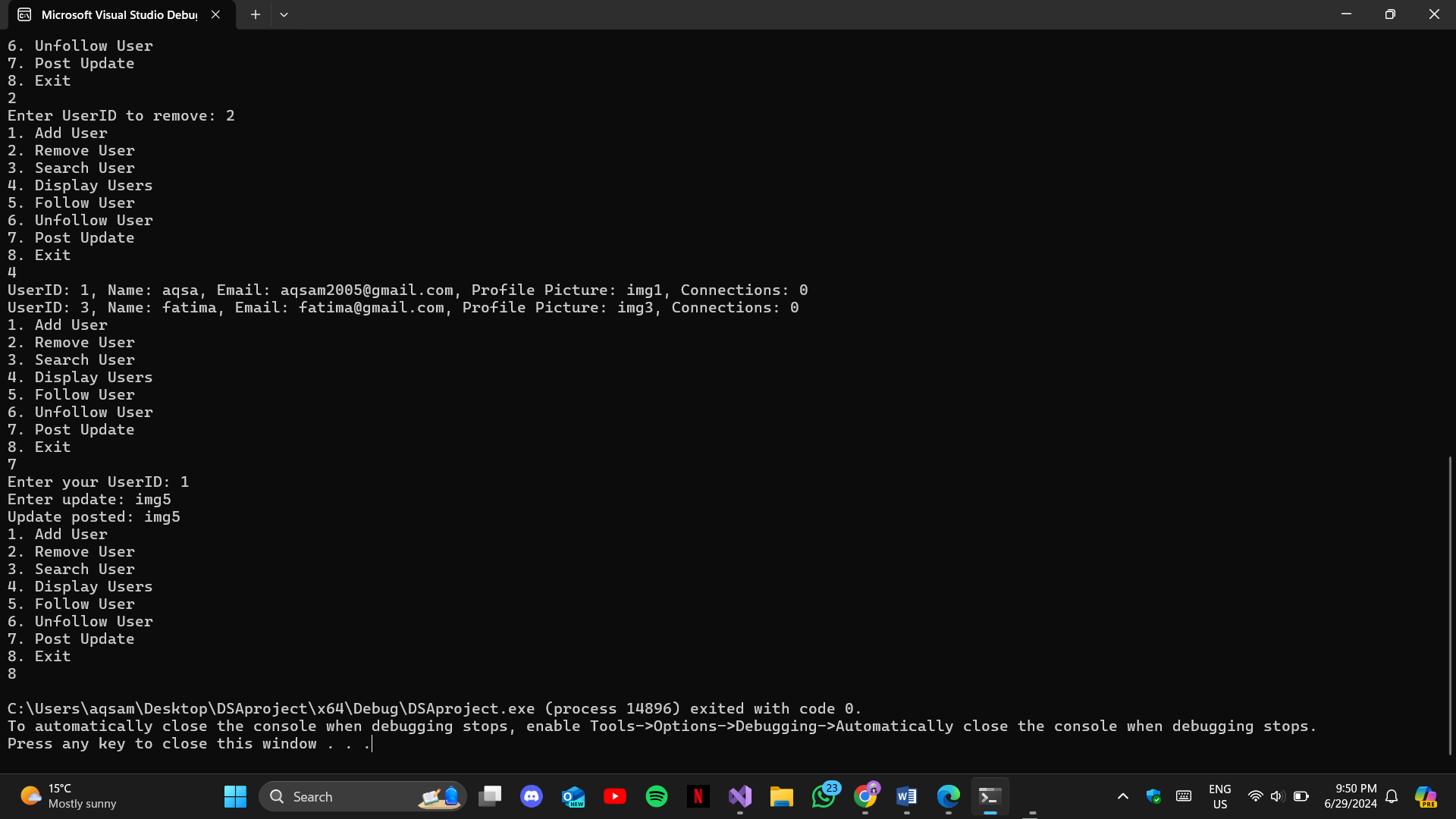
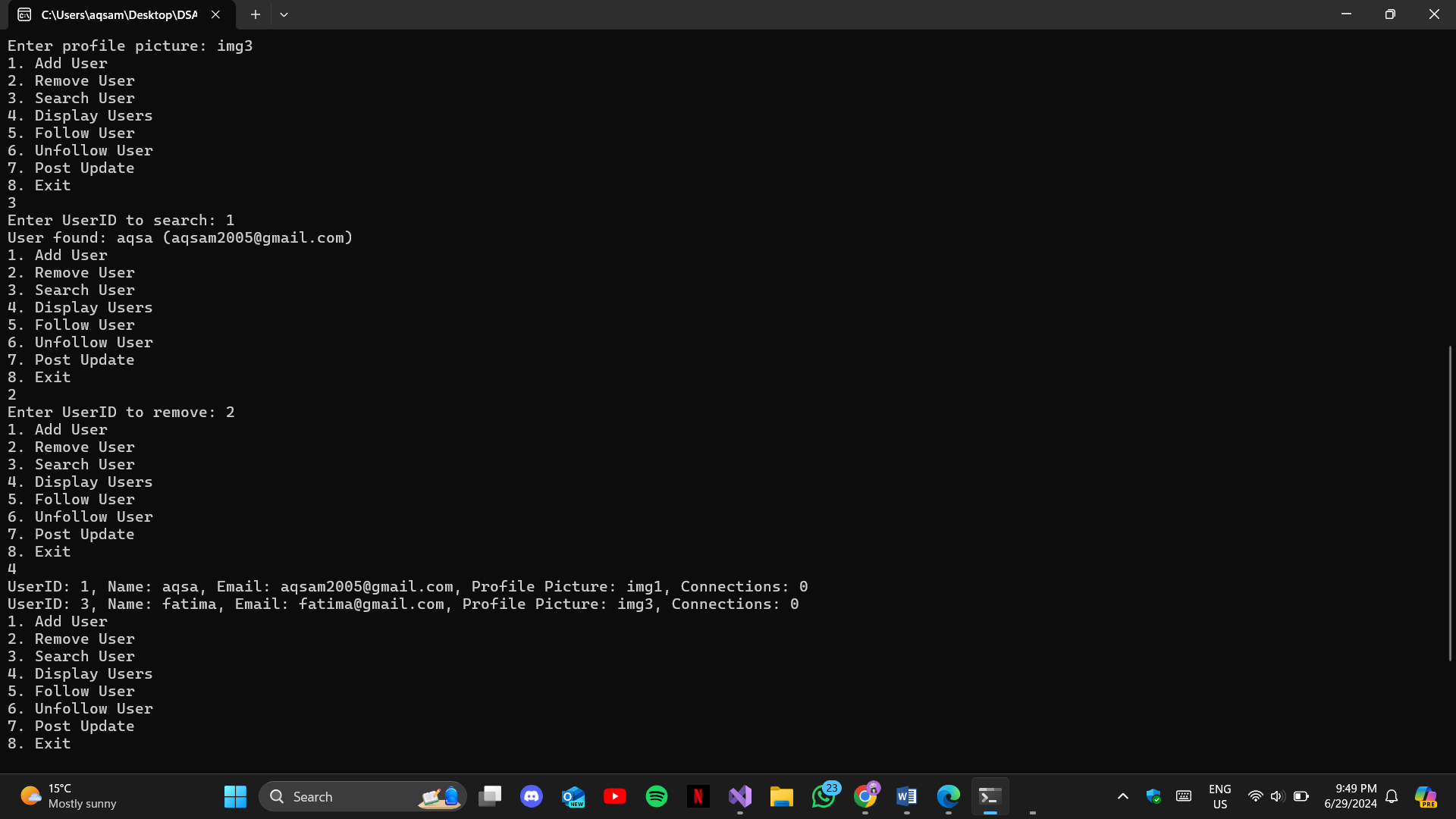
Successfully built the project



***Outputs:***

Here the few outputs from our project

******

******