

DATA Structures PROJECT

Zaid Ahmed 22F-3115



#include <iostream>

#include <string>

using namespace std;

const int MAX\_USERS = 100;

const int MAX\_REPOSITORIES = 100;

const int MAX\_FOLLOWERS = 100;

class User {

string username;

string password;

string followers[MAX\_USERS];

int num\_followers = 0;

bool loggedIn = false;

static int numUsers;

static User users[MAX\_USERS];

public:

bool registered;

User() {}

User(string username, string password)

{

this->username = username;

this->password = password;

registered = false;

}

void registerUser(const string& username, const string& password) {

int index = hash(username);

if (users[index].username.empty()) {

users[index].username = username;

users[index].password = password;

numUsers++;

cout << "User registered successfully!\n";

}

else {

cout << "Username already exists.\n";

}

registered = true;

}

bool isRegistered() {

return registered;

}

void login(const string& username, const string& password) {

int index = hash(username);

if (!users[index].username.empty() && users[index].password == password) {

users[index].loggedIn = true;

cout << "User logged in successfully!\n";

}

else {

cout << "Invalid username or password.\n";

}

}

void logout(const string& username) {

int index = hash(username);

if (!users[index].username.empty()) {

users[index].loggedIn = false;

cout << "User logged out successfully!\n";

}

else {

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

}

}

void viewProfile(const string& username) {

int index = hash(username);

if (!users[index].username.empty()) {

cout << "Username: " << users[index].username << endl;

cout << "Followers: " << users[index].num\_followers << endl;

}

else {

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

}

}

int hash(const string& username) {

int hashValue = 0;

for (char c : username) {

hashValue += c;

}

return hashValue % MAX\_USERS;

}

};

struct Commit {

string message;

Commit\* next;

Commit\* prev;

};

class Repository {

Commit\* headCommit;

Commit\* tailCommit;

int num\_repositories = 0;

public:

bool isPublic;

string name;

string owner;

Repository\* next;

int forkCount;

Repository\* head;

Repository()

{

head = NULL;

num\_repositories = 0;

}

void createRepository(const string& repoName, bool isPublic, const string& owner) {

if (num\_repositories < MAX\_REPOSITORIES) {

Repository\* newRepo = new Repository;

newRepo->name = repoName;

newRepo->isPublic = isPublic;

newRepo->owner = owner;

newRepo->forkCount = 0;

newRepo->next = NULL;

if (head == nullptr)

{

head = newRepo;

}

else

{

Repository\* current = head;

while (current->next != nullptr)

{

current = current->next;

}

current->next = newRepo;

}

num\_repositories++;

cout << "Repository created successfully!\n";

}

else

{

cout << "Maximum repository limit reached.\n";

}

}

void deleteRepository(const string& repoName, const string& username) {

Repository\* current = head;

Repository\* prev = nullptr;

while (current != nullptr) {

if (current->name == repoName && current->owner == username) {

Commit\* currentCommit = headCommit;

while (currentCommit != nullptr) {

Commit\* temp = currentCommit;

currentCommit = currentCommit->next;

delete temp;

}

if (prev == nullptr)

{

head = current->next;

}

else

{

prev->next = current->next;

}

delete current;

num\_repositories--;

cout << "Repository deleted successfully!\n";

return;

}

prev = current;

current = current->next;

}

cout << "Repository not found or you don't have permission to delete it.\n";

}

void forkRepository(const string& repoName, const string& username)

{

Repository\* originalRepo = findRepository(repoName);

if (originalRepo != nullptr && originalRepo->isPublic)

{

createRepository(repoName, originalRepo->isPublic, username);

originalRepo->forkCount++;

cout << "Repository forked successfully!\n";

}

else

{

cout << "Repository not found or cannot be forked.\n";

}

}

void setRepositoryVisibility(const string& repoName, bool isPublic, const string& username)

{

Repository\* repoToUpdate = findRepository(repoName);

if (repoToUpdate != nullptr && repoToUpdate->owner == username) {

repoToUpdate->isPublic = isPublic;

cout << "Repository visibility updated successfully!\n";

}

else {

cout << "Repository not found or you don't have permission to change visibility.\n";

}

}

void viewRepositoryStats(const string& repoName)

{

Repository\* repoToView = findRepository(repoName);

if (repoToView != nullptr)

{

cout << "Repository Name: " << repoToView->name << endl;

cout << "Repository Fork Count: " << repoToView->forkCount << endl;

}

else

{

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

}

}

Repository\* findRepository(const string& repoName) {

Repository\* current = head;

while (current != nullptr) {

if (current->name == repoName) {

return current;

}

current = current->next;

}

return nullptr;

}

};

class SocialFeatures {

string users[MAX\_USERS];

string followers[MAX\_USERS][MAX\_FOLLOWERS];

int numUsers;

public:

SocialFeatures() {

numUsers = 0;

}

int findUserIndex(const string& user) {

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

if (users[i] == user) {

return i;

}

}

return -1;

}

void followUser(const string& follower, const string& followee) {

int followerIndex = findUserIndex(follower);

int followeeIndex = findUserIndex(followee);

if (followerIndex != -1 && followeeIndex != -1) {

followers[followerIndex][numFollowers(followerIndex)++] = followee;

cout << follower << " is now following " << followee << ".\n";

}

else {

cout << "Invalid usernames. Please check and try again.\n";

}

}

void unfollowUser(const string& follower, const string& followee) {

int followerIndex = findUserIndex(follower);

int followeeIndex = findUserIndex(followee);

if (followerIndex != -1) {

int j = 0;

while (j < numFollowers(followerIndex)) {

if (followers[followerIndex][j] == followee) {

for (int k = j; k < numFollowers(followerIndex) - 1; ++k) {

followers[followerIndex][k] = followers[followerIndex][k + 1];

}

numFollowers(followerIndex)--;

cout << follower << " has unfollowed " << followee << ".\n";

return;

}

++j;

}

cout << follower << " is not following " << followee << ".\n";

}

else {

cout << "Invalid usernames. Please check and try again.\n";

}

}

int& numFollowers(int index) {

static int numFollowersArray[MAX\_USERS] = { 0 };

return numFollowersArray[index];

}

void printFollowers(const string& user) {

int userIndex = findUserIndex(user);

if (userIndex != -1) {

cout << user << " has the following followers:\n";

for (int i = 0; i < numFollowers(userIndex); ++i) {

cout << "- " << followers[userIndex][i] << "\n";

}

}

else {

cout << user << " is not following anyone.\n";

}

}

};

int User::numUsers = 0;

User User::users[MAX\_USERS];

int main() {

int choice;

string username, password, follower, followee, repoName;

User userObject;

Repository repoObject;

SocialFeatures socialObject;

bool loggedIn = false;

do {

cout << "GitHub Simulation Menu:\n";

cout << "1. Register User\n";

cout << "2. Login\n";

cout << "3. Logout\n";

cout << "4. Follow User\n";

cout << "5. Unfollow User\n";

cout << "6. View Profile\n";

cout << "7. Create Repository\n";

cout << "8. List User Repositories\n";

cout << "9. Delete Repository\n";

cout << "10. Fork Repository\n";

cout << "11. Set Repository Visibility\n";

cout << "12. View Repository Stats\n";

cout << "13. View Followers\n";

cout << "14. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1: {

cout << "Enter username: ";

cin >> username;

cout << "Enter password: ";

cin >> password;

userObject.registerUser(username, password);

system("pause");

system("cls");

break;

}

case 2: {

cout << "Enter username: ";

cin >> username;

cout << "Enter password: ";

cin >> password;

userObject.login(username, password);

loggedIn = userObject.isRegistered();

system("pause");

system("cls");

break;

}

case 3: {

cout << "Enter username: ";

cin >> username;

userObject.logout(username);

loggedIn = false;

system("pause");

system("cls");

break;

}

case 4: {

if (!loggedIn) {

cout << "Login first!\n";

system("pause");

system("cls");

break;

}

cout << "Enter follower username: ";

cin >> follower;

cout << "Enter followee username: ";

cin >> followee;

socialObject.followUser(follower, followee);

system("pause");

system("cls");

break;

}

case 5: {

if (!loggedIn) {

cout << "Login first!\n";

system("pause");

system("cls");

break;

}

cout << "Enter follower username: ";

cin >> follower;

cout << "Enter followee username: ";

cin >> followee;

socialObject.unfollowUser(follower, followee);

system("pause");

system("cls");

break;

}

case 6: {

cout << "Enter username: ";

cin >> username;

userObject.viewProfile(username);

system("pause");

system("cls");

break;

}

case 7: {

if (!loggedIn) {

cout << "Login first!\n";

system("pause");

system("cls");

break;

}

cout << "Enter repository name: ";

cin >> repoName;

cout << "Is the repository public? (1 for Yes, 0 for No): ";

bool isPublic;

cin >> isPublic;

repoObject.createRepository(repoName, isPublic, username);

system("pause");

system("cls");

break;

}

case 8: {

if (!loggedIn) {

cout << "Login first!\n";

system("pause");

system("cls");

break;

}

cout << "Enter username: ";

cin >> username;

Repository\* currentRepo = repoObject.head;

bool found = false;

while (currentRepo != nullptr) {

if (currentRepo->owner == username) {

cout << "Repository Name: " << currentRepo->name << ", Visibility: " << (currentRepo->isPublic ? "Public" : "Private") << endl;

found = true;

}

currentRepo = currentRepo->next;

}

if (!found) {

cout << "No repositories found for user " << username << ".\n";

}

system("pause");

system("cls");

break;

}

case 9: {

if (!loggedIn) {

cout << "Login first!\n";

system("pause");

system("cls");

break;

}

cout << "Enter repository name: ";

cin >> repoName;

repoObject.deleteRepository(repoName, username);

system("pause");

system("cls");

break;

}

case 10: {

if (!loggedIn) {

cout << "Login first!\n";

system("pause");

system("cls");

break;

}

cout << "Enter repository name: ";

cin >> repoName;

repoObject.forkRepository(repoName, username);

system("pause");

system("cls");

break;

}

case 11: {

if (!loggedIn) {

cout << "Login first!\n";

system("pause");

system("cls");

break;

}

cout << "Enter repository name: ";

cin >> repoName;

cout << "Is the repository public? (1 for Yes, 0 for No): ";

bool isPublic;

cin >> isPublic;

repoObject.setRepositoryVisibility(repoName, isPublic, username);

system("pause");

system("cls");

break;

}

case 12: {

if (!loggedIn) {

cout << "Login first!\n";

system("pause");

system("cls");

break;

}

cout << "Enter repository name: ";

cin >> repoName;

repoObject.viewRepositoryStats(repoName);

system("pause");

system("cls");

break;

}

case 13: {

cout << "Enter username: ";

cin >> username;

socialObject.printFollowers(username);

system("pause");

system("cls");

break;

}

case 14: {

cout << "Exiting. Goodbye!\n";

system("pause");

system("cls");

return 0;

}

default:

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

system("pause");

system("cls");

}

} while (choice != 14);

return 0;

}

