```cpp
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <ctime>
#include <algorithm>
#include <sstream>
#include <cctype>
#include <limits>

using namespace std;

// Helper functions
vector<string> splitString(const string &s, char delimiter) {
    vector<string> tokens;
    string token;
    istringstream tokenStream(s);
    while (getline(tokenStream, token, delimiter)) {
        tokens.push_back(token);
    }
    return tokens;
}

string join(const vector<string>& v, char delimiter) {
    string s;
    for (size_t i = 0; i < v.size(); ++i) {
        if (i != 0) {
            s += delimiter;
        }
        s += v[i];
    }
    return s;
}

string trim(const string &s) {
    auto start = s.begin();
    while (start != s.end() && isspace(*start)) {
        start++;
    }
    auto end = s.end();
    do {
        end--;
    } while (distance(start, end) > 0 && isspace(*end));
    return string(start, end + 1);
}

string getCurrentTimestamp() {
    time_t now = time(0);
    string timestamp = ctime(&now);
    timestamp.erase(remove(timestamp.begin(), timestamp.end(), '\n'), timestamp.end());
    return timestamp;
}

void clearInputBuffer() {
    cin.clear();
    cin.ignore(numeric_limits<streamsize>::max(), '\n');
}

// Data structures
struct User {
    string username;
    string email;
    string password;
    bool isAdmin;
};

```

```cpp
struct Post {
    int postId;
    string title;
    string content;
    string author;
    string timestamp;
    vector<string> tags;
};

struct Comment {
    int commentId;
    int postId;
    string author;
    string content;
    string timestamp;
};

// Class declarations
class UserManager {
private:
    string currentUser;
    bool isAdmin;
    string userFile;

    bool usernameExists(const string &username) {
        ifstream file(userFile);
        string line;
        while (getline(file, line)) {
            vector<string> parts = splitString(line, '|');
            if (!parts.empty() && parts[0] == username) {
                return true;
            }
        }
        return false;
    }

    bool isValidEmail(const string &email) {
        const string domain = "@mail.jiit.ac.in";
        return email.size() > domain.size() &&
               email.substr(email.size() - domain.size()) == domain;
    }

    bool isPasswordStrong(const string &password) {
        if (password.length() < 8) {
            return false;
        }
        bool hasUpper = false, hasLower = false, hasDigit = false, hasSymbol = false;
        for (char c : password) {
            if (isupper(c)) {
                hasUpper = true;
            }
            else if (islower(c)) {
                hasLower = true;
            }
            else if (isdigit(c)) {
                hasDigit = true;
            }
            else {
                hasSymbol = true;
            }
        }
        return hasUpper && hasLower && hasDigit && hasSymbol;
    }

public:
    UserManager() : userFile("users.txt"), currentUser(""), isAdmin(false) {}
```

```
133
134     bool registerUser() {
135         User newUser;
136         cout << "Enter username: ";
137         getline(cin, newUser.username);
138         newUser.username = trim(newUser.username);
139
140         if (newUser.username.empty()) {
141             cout << "Username cannot be empty!\n";
142             return false;
143         }
144
145         if (usernameExists(newUser.username)) {
146             cout << "Username already exists!\n";
147             return false;
148         }
149
150         cout << "Enter email: ";
151         getline(cin, newUser.email);
152         newUser.email = trim(newUser.email);
153
154         if (!isValidEmail(newUser.email)) {
155             cout << "Invalid email domain! Use @mail.jiit.ac.in\n";
156             return false;
157         }
158
159         cout << "Enter password: ";
160         getline(cin, newUser.password);
161
162         if (!isPasswordStrong(newUser.password)) {
163             cout << "Password must be 8 characters or more, and must contain atleast one uppercase,\none
lowercase, one number and one symbol.\n";
164             return false;
165         }
166
167         newUser.isAdmin = false;
168         ofstream file(userFile, ios::app);
169         file << newUser.username << "|" << newUser.email << "|" << newUser.password << "|0\n";
170         cout << "Registration successful!\n";
171         return true;
172     }
173
174     bool login() {
175         string username, password;
176         cout << "Username: ";
177         getline(cin, username);
178         cout << "Password: ";
179         getline(cin, password);
180
181         ifstream file(userFile);
182         string line;
183         while (getline(file, line)) {
184             vector<string> parts = splitString(line, '|');
185             if (parts.size() >= 4 && parts[0] == username && parts[2] == password) {
186                 currentUser = username;
187                 isAdmin = (parts[3] == "1");
188                 return true;
189             }
190         }
191         cout << "Invalid credentials!\n";
192         return false;
193     }
194
195     void logout() {
196         currentUser = "";
197         isAdmin = false;
```

```
    }

    string getCurrentUser() { return currentUser; }
    bool isUserAdmin() { return isAdmin; }
    bool isLoggedIn() { return !currentUser.empty(); }

    vector<string> getAllUsernames() {
        vector<string> usernames;
        ifstream file(userFile);
        string line;
        while (getline(file, line)) {
            vector<string> parts = splitString(line, '|');
            if (!parts.empty()) {
                usernames.push_back(parts[0]);
            }
        }
        return usernames;
    }
};

class BlogManager {
private:
    string postFile;
    string commentFile;

    int generatePostId() {
        ifstream file(postFile);
        string line;
        int maxId = 0;
        while (getline(file, line)) {
            vector<string> parts = splitString(line, '|');
            if (!parts.empty()) {
                int id = stoi(parts[0]);
                if (id > maxId) {
                    maxId = id;
                }
            }
        }
        return maxId + 1;
    }

    int generateCommentId() {
        ifstream file(commentFile);
        string line;
        int maxId = 0;
        while (getline(file, line)) {
            vector<string> parts = splitString(line, '|');
            if (!parts.empty()) {
                int id = stoi(parts[0]);
                if (id > maxId) {
                    maxId = id;
                }
            }
        }
        return maxId + 1;
    }

    Post parsePost(const vector<string>& parts) {
        Post post;
        post.postId = stoi(parts[0]);
        post.title = parts[1];
        post.content = parts[2];
        post.author = parts[3];
        post.timestamp = parts[4];
        if (parts.size() > 5) {
            post.tags = splitString(parts[5], ',');
```

```
        }
        return post;
    }

    Comment parseComment(const vector<string>& parts) {
        Comment comment;
        comment.commentId = stoi(parts[0]);
        comment.postId = stoi(parts[1]);
        comment.author = parts[2];
        comment.content = parts[3];
        comment.timestamp = parts[4];
        return comment;
    }

public:
    BlogManager() : postFile("posts.txt"), commentFile("comments.txt") {}

    void createPost(const string &author) {
        Post newPost;
        newPost.postId = generatePostId();
        cout << "Enter post title: ";
        getline(cin, newPost.title);
        cout << "Enter post content: ";
        getline(cin, newPost.content);

        cout << "Enter tags (comma-separated): ";
        string tagsInput;
        getline(cin, tagsInput);
        newPost.tags = splitString(tagsInput, ',');
        // Trim whitespace from tags
        for (auto& tag : newPost.tags) {
            tag = trim(tag);
        }

        newPost.timestamp = getCurrentTimestamp();
        newPost.author = author;

        ofstream file(postFile, ios::app);
        file << newPost.postId << "|" << newPost.title << "|" << newPost.content << "|"
             << newPost.author << "|" << newPost.timestamp << "|" << join(newPost.tags, ',') << endl;
        cout << "Post created successfully with ID: " << newPost.postId << "!\n";
    }

    bool editPost(int postId, const string &currentUser) {
        ifstream inFile(postFile);
        ofstream outFile("temp.txt");
        bool found = false;
        string line;
        Post postToEdit;

        while (getline(inFile, line)) {
            vector<string> parts = splitString(line, '|');
            if (!parts.empty() && stoi(parts[0]) == postId) {
                if (parts[3] == currentUser) {
                    found = true;
                    postToEdit = parsePost(parts);

                    cout << "Current title: " << postToEdit.title << "\nEnter new title (or press enter to
keep current): ";
                    string newTitle;
                    getline(cin, newTitle);
                    if (!newTitle.empty()) {
                        postToEdit.title = newTitle;
                    }

                    cout << "Current content: " << postToEdit.content << "\nEnter new content (or press
```

```
enter to keep current): ";
                    string newContent;
                    getline(cin, newContent);
                    if (!newContent.empty()) {
                        postToEdit.content = newContent;
                    }

                    cout << "Current tags: " << join(postToEdit.tags, ',') << "\nEnter new tags
(comma-separated, or press enter to keep current): ";
                    string newTags;
                    getline(cin, newTags);
                    if (!newTags.empty()) {
                        postToEdit.tags = splitString(newTags, ',');
                        for (auto& tag : postToEdit.tags) {
                            tag = trim(tag);
                        }
                    }

                    postToEdit.timestamp = getCurrentTimestamp();

                    outFile << postToEdit.postId << "|" << postToEdit.title << "|" << postToEdit.content <<
"|"
                            << postToEdit.author << "|" << postToEdit.timestamp << "|" << join(postToEdit.
tags, ',') << endl;
                    continue;
                }
            }
            outFile << line << endl;
        }

        inFile.close();
        outFile.close();
        remove(postFile.c_str());
        rename("temp.txt", postFile.c_str());

        if (found) {
            cout << "Post updated successfully!\n";
        }
        else {
            cout << "Post not found or you don't have permission to edit it!\n";
        }
        return found;
    }

    vector<Post> getUserPosts(const string &username) {
        vector<Post> posts;
        ifstream file(postFile);
        string line;
        while (getline(file, line)) {
            vector<string> parts = splitString(line, '|');
            if (parts.size() >= 6 && parts[3] == username) {
                posts.push_back(parsePost(parts));
            }
        }
        return posts;
    }

    vector<Post> getAllPosts() {
        vector<Post> posts;
        ifstream file(postFile);
        string line;
        while (getline(file, line)) {
            vector<string> parts = splitString(line, '|');
            if (parts.size() >= 6) {
                posts.push_back(parsePost(parts));
            }
```

```
        }
        return posts;
    }

    vector<Post> searchPostsByTag(const string &tag) {
        vector<Post> results;
        ifstream file(postFile);
        string line;
        while (getline(file, line)) {
            vector<string> parts = splitString(line, '|');
            if (parts.size() >= 6) {
                Post post = parsePost(parts);
                for (const auto &t : post.tags) {
                    if (t == tag) {
                        results.push_back(post);
                        break;
                    }
                }
            }
        }
        return results;
    }

    vector<Post> searchPostsByUsername(const string &username) {
        return getUserPosts(username);
    }

    void displayPosts(const vector<Post> &posts) {
        if (posts.empty()) {
            cout << "No posts found.\n";
            return;
        }
        for (const auto &post : posts) {
            cout << "\nID: " << post.postId << "\nTitle: " << post.title
                 << "\nContent: " << post.content << "\nAuthor: " << post.author
                 << "\nTags: " << join(post.tags, ',') << "\nDate: " << post.timestamp << "\n";
        }
    }

    bool deletePost(int postId, const string &currentUser, bool isAdmin) {
        ifstream inFile(postFile);
        ofstream outFile("temp.txt");
        bool found = false;
        string line;

        while (getline(inFile, line)) {
            vector<string> parts = splitString(line, '|');
            if (!parts.empty() && stoi(parts[0]) == postId) {
                if (parts[3] == currentUser || isAdmin) {
                    found = true;
                    continue;
                }
            }
            outFile << line << endl;
        }

        inFile.close();
        outFile.close();
        remove(postFile.c_str());
        rename("temp.txt", postFile.c_str());

        // Also delete associated comments
        if (found) {
            deleteCommentsForPost(postId);
        }
        return found;
```

```
}

void addComment(int postId, const string &author) {
    Comment newComment;
    newComment.commentId = generateCommentId();
    newComment.postId = postId;
    newComment.author = author;
    cout << "Enter your comment: ";
    getline(cin, newComment.content);
    newComment.timestamp = getCurrentTimestamp();

    ofstream file(commentFile, ios::app);
    file << newComment.commentId << "|" << newComment.postId << "|"
         << newComment.author << "|" << newComment.content << "|"
         << newComment.timestamp << endl;
    cout << "Comment added successfully!\n";
}

vector<Comment> getCommentsForPost(int postId) {
    vector<Comment> comments;
    ifstream file(commentFile);
    string line;
    while (getline(file, line)) {
        vector<string> parts = splitString(line, '|');
        if (parts.size() >= 5 && stoi(parts[1]) == postId) {
            comments.push_back(parseComment(parts));
        }
    }
    return comments;
}

void displayComments(const vector<Comment> &comments) {
    if (comments.empty()) {
        cout << "No comments yet.\n";
        return;
    }
    for (const auto &comment : comments) {
        cout << "\nID: " << comment.commentId << "\nAuthor: " << comment.author
             << "\nComment: " << comment.content << "\nDate: " << comment.timestamp << "\n";
    }
}

bool deleteComment(int commentId, const string &currentUser, bool isAdmin) {
    ifstream inFile(commentFile);
    ofstream outFile("temp.txt");
    bool found = false;
    string line;

    while (getline(inFile, line)) {
        vector<string> parts = splitString(line, '|');
        if (!parts.empty() && stoi(parts[0]) == commentId) {
            if (parts[2] == currentUser || isAdmin) {
                found = true;
                continue;
            }
        }
        outFile << line << endl;
    }

    inFile.close();
    outFile.close();
    remove(commentFile.c_str());
    rename("temp.txt", commentFile.c_str());
    return found;
}
```

```
    bool editComment(int commentId, const string &currentUser) {
        ifstream inFile(commentFile);
        ofstream outFile("temp.txt");
        bool found = false;
        string line;
        Comment commentToEdit;

        while (getline(inFile, line)) {
            vector<string> parts = splitString(line, '|');
            if (!parts.empty() && stoi(parts[0]) == commentId) {
                if (parts[2] == currentUser) {
                    found = true;
                    commentToEdit = parseComment(parts);

                    cout << "Current comment: " << commentToEdit.content << "\nEnter new comment: ";
                    string newContent;
                    getline(cin, newContent);
                    if (!newContent.empty()) {
                        commentToEdit.content = newContent;
                    }

                    commentToEdit.timestamp = getCurrentTimestamp();

                    outFile << commentToEdit.commentId << "|" << commentToEdit.postId << "|"
                            << commentToEdit.author << "|" << commentToEdit.content << "|"
                            << commentToEdit.timestamp << endl;
                    continue;
                }
            }
            outFile << line << endl;
        }

        inFile.close();
        outFile.close();
        remove(commentFile.c_str());
        rename("temp.txt", commentFile.c_str());

        if (found) {
            cout << "Comment updated successfully!\n";
        }
        else {
            cout << "Comment not found or you don't have permission to edit it!\n";
        }
        return found;
    }

private:
    void deleteCommentsForPost(int postId) {
        ifstream inFile(commentFile);
        ofstream outFile("temp.txt");
        string line;

        while (getline(inFile, line)) {
            vector<string> parts = splitString(line, '|');
            if (!parts.empty() && stoi(parts[1]) != postId) {
                outFile << line << endl;
            }
        }

        inFile.close();
        outFile.close();
        remove(commentFile.c_str());
        rename("temp.txt", commentFile.c_str());
    }
};

```

```
class AdminManager {
private:
    UserManager* userManager;
    BlogManager* blogManager;

    void deleteUser() {
        string username;
        cout << "Enter username to delete: ";
        getline(cin, username);

        if (username == userManager->getCurrentUser()) {
            cout << "You cannot delete yourself!\n";
            return;
        }

        ifstream inFile("users.txt");
        ofstream outFile("temp.txt");
        string line;
        bool found = false;

        while (getline(inFile, line)) {
            vector<string> parts = splitString(line, '|');
            if (!parts.empty() && parts[0] == username) {
                found = true;
                continue;
            }
            outFile << line << endl;
        }

        inFile.close();
        outFile.close();
        remove("users.txt");
        rename("temp.txt", "users.txt");

        if (found) {
            cout << "User deleted successfully!\n";
            // Also delete user's posts and comments
            vector<Post> userPosts = blogManager->getUserPosts(username);
            for (const auto& post : userPosts) {
                blogManager->deletePost(post.postId, username, true);
            }
        }
        else {
            cout << "User not found!\n";
        }
    }

    void makeAdmin() {
        string username;
        cout << "Enter username to make admin: ";
        getline(cin, username);

        ifstream inFile("users.txt");
        ofstream outFile("temp.txt");
        string line;
        bool found = false;

        while (getline(inFile, line)) {
            vector<string> parts = splitString(line, '|');
            if (!parts.empty() && parts[0] == username) {
                found = true;
                parts[3] = "1";
                outFile << join(parts, '|') << endl;
            }
            else {
                outFile << line << endl;
```

```
            }
        }

        inFile.close();
        outFile.close();
        remove("users.txt");
        rename("temp.txt", "users.txt");

        if (found) {
            cout << "User is now an admin!\n";
        }
        else {
            cout << "User not found!\n";
        }
    }

    void deleteAnyPost() {
        int postId;
        cout << "Enter post ID to delete: ";
        cin >> postId;
        clearInputBuffer();

        if (blogManager->deletePost(postId, "", true)) {
            cout << "Post deleted successfully!\n";
        }
        else {
            cout << "Post not found!\n";
        }
    }

public:
    AdminManager(UserManager* um, BlogManager* bm) : userManager(um), blogManager(bm) {}

    void adminMenu() {
        while (true) {
            cout << "\nAdmin Panel\n1. Delete User\n2. Make User Admin\n3. Delete Any Post\n4.
Back\nChoice: ";
            int choice;
            cin >> choice;
            clearInputBuffer();

            switch (choice) {
                case 1:
                    deleteUser();
                    break;
                case 2:
                    makeAdmin();
                    break;
                case 3:
                    deleteAnyPost();
                    break;
                case 4:
                    return;
                default:
                    cout << "Invalid choice!\n";
            }
        }
    }
};

class TagManager {
private:
    BlogManager* blogManager;

public:
    TagManager(BlogManager* bm) : blogManager(bm) {}
```

```
720
721     void manageTags(const string &currentUser) {
722         vector<Post> userPosts = blogManager->getUserPosts(currentUser);
723         if (userPosts.empty()) {
724             cout << "You have no posts to manage tags for.\n";
725             return;
726         }
727
728         cout << "Your posts:\n";
729         blogManager->displayPosts(userPosts);
730
731         int postId;
732         cout << "Enter post ID to manage tags: ";
733         cin >> postId;
734         clearInputBuffer();
735
736         bool found = false;
737         Post targetPost;
738         for (const auto& post : userPosts) {
739             if (post.postId == postId) {
740                 found = true;
741                 targetPost = post;
742                 break;
743             }
744         }
745
746         if (!found) {
747             cout << "Post not found or you don't have permission to edit it.\n";
748             return;
749         }
750
751         while (true) {
752             cout << "\nCurrent tags: " << join(targetPost.tags, ',') << "\n";
753             cout << "1. Add tag\n2. Delete tag\n3. Back\nChoice: ";
754             int choice;
755             cin >> choice;
756             clearInputBuffer();
757
758             if (choice == 1) {
759                 cout << "Enter new tag to add: ";
760                 string newTag;
761                 getline(cin, newTag);
762                 newTag = trim(newTag);
763                 if (!newTag.empty()) {
764                     if (find(targetPost.tags.begin(), targetPost.tags.end(), newTag) == targetPost.tags.end
()) {
765                         targetPost.tags.push_back(newTag);
766                         blogManager->editPost(postId, currentUser); // This will trigger a reload
767                         cout << "Tag added successfully!\n";
768                     }
769                     else {
770                         cout << "Tag already exists!\n";
771                     }
772                 }
773             }
774             else if (choice == 2) {
775                 cout << "Enter tag to delete: ";
776                 string tagToDelete;
777                 getline(cin, tagToDelete);
778                 tagToDelete = trim(tagToDelete);
779
780                 auto it = find(targetPost.tags.begin(), targetPost.tags.end(), tagToDelete);
781                 if (it != targetPost.tags.end()) {
782                     targetPost.tags.erase(it);
783                     blogManager->editPost(postId, currentUser); // This will trigger a reload
784                     cout << "Tag deleted successfully!\n";
```

```
                }
                else {
                    cout << "Tag not found!\n";
                }
            }
            else if (choice == 3) {
                break;
            }
            else {
                cout << "Invalid choice!\n";
            }
        }
    }
};

// Main application
int main() {
    UserManager userManager;
    BlogManager blogManager;
    AdminManager adminManager(&userManager, &blogManager);
    TagManager tagManager(&blogManager);

    cout<<"Welcome to USER BLOG MANAGEMENT SYSTEM!"<<endl;

    while (true) {
        if (!userManager.isLoggedIn()) {
            cout << "\nMain Menu\n1. Register\n2. Login\n3. Exit\nChoice: ";
            int choice;
            cin >> choice;
            clearInputBuffer();

            if (choice == 1) {
                userManager.registerUser();
            }
            else if (choice == 2) {
                userManager.login();
            }
            else if (choice == 3) {
                break;
            }
            else {
                cout << "Invalid choice!\n";
            }
        }
        else {
            cout << "\nWelcome " << userManager.getCurrentUser() << "!\n";
            if (userManager.isUserAdmin()) {
                cout << "1. Admin Panel\n";
            }
            cout << "1. Create Post\n2. My Posts\n3. Edit Post\n4. Delete Post\n5. Manage Tags\n"
                 << "6. Search Posts by Tag\n7. Search Posts by Username\n8. View All Posts\n"
                 << "9. Add Comment\n10. View/Edit/Delete Comments\n11. Logout\nChoice: ";

            int choice;
            cin >> choice;
            clearInputBuffer();

            if (userManager.isUserAdmin() && choice == 1) {
                adminManager.adminMenu();
            }
            else {
                switch (choice) {
                    case 1:
                        blogManager.createPost(userManager.getCurrentUser());
                        break;
                    case 2: {
```

```
                    vector<Post> posts = blogManager.getUserPosts(userManager.getCurrentUser());
                    blogManager.displayPosts(posts);
                    break;
                }
                case 3: {
                    int postId;
                    cout << "Enter post ID to edit: ";
                    cin >> postId;
                    clearInputBuffer();
                    blogManager.editPost(postId, userManager.getCurrentUser());
                    break;
                }
                case 4: {
                    int postId;
                    cout << "Enter post ID to delete: ";
                    cin >> postId;
                    clearInputBuffer();
                    if (blogManager.deletePost(postId, userManager.getCurrentUser(), false)) {
                        cout << "Post deleted!\n";
                    }
                    else {
                        cout << "Post not found or unauthorized!\n";
                    }
                    break;
                }
                case 5:
                    tagManager.manageTags(userManager.getCurrentUser());
                    break;
                case 6: {
                    string tag;
                    cout << "Enter tag to search: ";
                    getline(cin, tag);
                    vector<Post> results = blogManager.searchPostsByTag(tag);
                    blogManager.displayPosts(results);
                    break;
                }
                case 7: {
                    string username;
                    cout << "Enter username to search: ";
                    getline(cin, username);
                    vector<Post> results = blogManager.searchPostsByUsername(username);
                    blogManager.displayPosts(results);
                    break;
                }
                case 8: {
                    vector<Post> allPosts = blogManager.getAllPosts();
                    blogManager.displayPosts(allPosts);
                    break;
                }
                case 9: {
                    int postId;
                    cout << "Enter post ID to comment on: ";
                    cin >> postId;
                    clearInputBuffer();
                    blogManager.addComment(postId, userManager.getCurrentUser());
                    break;
                }
                case 10: {
                    int postId;
                    cout << "Enter post ID to view comments: ";
                    cin >> postId;
                    clearInputBuffer();

                    vector<Comment> comments = blogManager.getCommentsForPost(postId);
                    blogManager.displayComments(comments);

```

```
                        if (!comments.empty()) {
                            cout << "\n1. Edit Comment\n2. Delete Comment\n3. Back\nChoice: ";
                            int commentChoice;
                            cin >> commentChoice;
                            clearInputBuffer();

                            if (commentChoice == 1 || commentChoice == 2) {
                                int commentId;
                                cout << "Enter comment ID: ";
                                cin >> commentId;
                                clearInputBuffer();

                                if (commentChoice == 1) {
                                    blogManager.editComment(commentId, userManager.getCurrentUser());
                                }
                                else {
                                    if (blogManager.deleteComment(commentId, userManager.getCurrentUser(),
false)) {
                                        cout << "Comment deleted!\n";
                                    }
                                    else {
                                        cout << "Comment not found or unauthorized!\n";
                                    }
                                }
                            }
                        }
                        break;
                    }
                    case 11:
                        userManager.logout();
                        break;
                    default:
                        cout << "Invalid choice!\n";
                }
            }
        }
    }
    return 0;

```