

```

1  #include <iostream>
2  #include <fstream>
3  #include <vector>
4  #include <string>
5  #include <ctime>
6  #include <algorithm>
7  #include <sstream>
8  #include <cctype>
9  #include <limits>
10
11 using namespace std;
12
13 // Helper functions
14 vector<string> splitString(const string &s, char delimiter) {
15     vector<string> tokens;
16     string token;
17     istringstream tokenStream(s);
18     while (getline(tokenStream, token, delimiter)) {
19         tokens.push_back(token);
20     }
21     return tokens;
22 }
23
24 string join(const vector<string>& v, char delimiter) {
25     string s;
26     for (size_t i = 0; i < v.size(); ++i) {
27         if (i != 0) {
28             s += delimiter;
29         }
30         s += v[i];
31     }
32     return s;
33 }
34
35 string trim(const string &s) {
36     auto start = s.begin();
37     while (start != s.end() && isspace(*start)) {
38         start++;
39     }
40     auto end = s.end();
41     do {
42         end--;
43     } while (distance(start, end) > 0 && isspace(*end));
44     return string(start, end + 1);
45 }
46
47 string getCurrentTimestamp() {
48     time_t now = time(0);
49     string timestamp = ctime(&now);
50     timestamp.erase(remove(timestamp.begin(), timestamp.end(), '\n'), timestamp.end());
51     return timestamp;
52 }
53
54 void clearInputBuffer() {
55     cin.clear();
56     cin.ignore(numeric_limits<streamsize>::max(), '\n');
57 }
58
59 // Data structures
60 struct User {
61     string username;
62     string email;
63     string password;
64     bool isAdmin;
65 };
66

```

```

67 struct Post {
68     int postId;
69     string title;
70     string content;
71     string author;
72     string timestamp;
73     vector<string> tags;
74 };
75
76 struct Comment {
77     int commentId;
78     int postId;
79     string author;
80     string content;
81     string timestamp;
82 };
83
84 // Class declarations
85 class UserManager {
86 private:
87     string currentUser;
88     bool isAdmin;
89     string userFile;
90
91     bool usernameExists(const string &username) {
92         ifstream file(userFile);
93         string line;
94         while (getline(file, line)) {
95             vector<string> parts = splitString(line, '|');
96             if (!parts.empty() && parts[0] == username) {
97                 return true;
98             }
99         }
100         return false;
101     }
102
103     bool isValidEmail(const string &email) {
104         const string domain = "@mail.jiit.ac.in";
105         return email.size() > domain.size() &&
106             email.substr(email.size() - domain.size()) == domain;
107     }
108
109     bool isPasswordStrong(const string &password) {
110         if (password.length() < 8) {
111             return false;
112         }
113         bool hasUpper = false, hasLower = false, hasDigit = false, hasSymbol = false;
114         for (char c : password) {
115             if (isupper(c)) {
116                 hasUpper = true;
117             }
118             else if (islower(c)) {
119                 hasLower = true;
120             }
121             else if (isdigit(c)) {
122                 hasDigit = true;
123             }
124             else {
125                 hasSymbol = true;
126             }
127         }
128         return hasUpper && hasLower && hasDigit && hasSymbol;
129     }
130
131 public:
132     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;

```

```

198     }
199
200     string getCurrentUser() { return currentUser; }
201     bool isUserAdmin() { return isAdmin; }
202     bool isLoggedIn() { return !currentUser.empty(); }
203
204     vector<string> getAllUsernames() {
205         vector<string> usernames;
206         ifstream file(userFile);
207         string line;
208         while (getline(file, line)) {
209             vector<string> parts = splitString(line, '|');
210             if (!parts.empty()) {
211                 usernames.push_back(parts[0]);
212             }
213         }
214         return usernames;
215     }
216 };
217
218 class BlogManager {
219 private:
220     string postFile;
221     string commentFile;
222
223     int generatePostId() {
224         ifstream file(postFile);
225         string line;
226         int maxId = 0;
227         while (getline(file, line)) {
228             vector<string> parts = splitString(line, '|');
229             if (!parts.empty()) {
230                 int id = stoi(parts[0]);
231                 if (id > maxId) {
232                     maxId = id;
233                 }
234             }
235         }
236         return maxId + 1;
237     }
238
239     int generateCommentId() {
240         ifstream file(commentFile);
241         string line;
242         int maxId = 0;
243         while (getline(file, line)) {
244             vector<string> parts = splitString(line, '|');
245             if (!parts.empty()) {
246                 int id = stoi(parts[0]);
247                 if (id > maxId) {
248                     maxId = id;
249                 }
250             }
251         }
252         return maxId + 1;
253     }
254
255     Post parsePost(const vector<string>& parts) {
256         Post post;
257         post.postId = stoi(parts[0]);
258         post.title = parts[1];
259         post.content = parts[2];
260         post.author = parts[3];
261         post.timestamp = parts[4];
262         if (parts.size() > 5) {
263             post.tags = splitString(parts[5], ',');

```

```

264     }
265     return post;
266 }
267
268 Comment parseComment(const vector<string>& parts) {
269     Comment comment;
270     comment.commentId = stoi(parts[0]);
271     comment.postId = stoi(parts[1]);
272     comment.author = parts[2];
273     comment.content = parts[3];
274     comment.timestamp = parts[4];
275     return comment;
276 }
277
278 public:
279     BlogManager() : postFile("posts.txt"), commentFile("comments.txt") {}
280
281     void createPost(const string &author) {
282         Post newPost;
283         newPost.postId = generatePostId();
284         cout << "Enter post title: ";
285         getline(cin, newPost.title);
286         cout << "Enter post content: ";
287         getline(cin, newPost.content);
288
289         cout << "Enter tags (comma-separated): ";
290         string tagsInput;
291         getline(cin, tagsInput);
292         newPost.tags = splitString(tagsInput, ',');
293         // Trim whitespace from tags
294         for (auto& tag : newPost.tags) {
295             tag = trim(tag);
296         }
297
298         newPost.timestamp = getCurrentTimestamp();
299         newPost.author = author;
300
301         ofstream file(postFile, ios::app);
302         file << newPost.postId << "|" << newPost.title << "|" << newPost.content << "|"
303             << newPost.author << "|" << newPost.timestamp << "|" << join(newPost.tags, ',') << endl;
304         cout << "Post created successfully with ID: " << newPost.postId << "!\n";
305     }
306
307     bool editPost(int postId, const string &currentUser) {
308         ifstream inFile(postFile);
309         ofstream outFile("temp.txt");
310         bool found = false;
311         string line;
312         Post postToEdit;
313
314         while (getline(inFile, line)) {
315             vector<string> parts = splitString(line, '|');
316             if (!parts.empty() && stoi(parts[0]) == postId) {
317                 if (parts[3] == currentUser) {
318                     found = true;
319                     postToEdit = parsePost(parts);
320
321                     cout << "Current title: " << postToEdit.title << "\nEnter new title (or press enter to
keep current): ";
322                     string newTitle;
323                     getline(cin, newTitle);
324                     if (!newTitle.empty()) {
325                         postToEdit.title = newTitle;
326                     }
327
328                     cout << "Current content: " << postToEdit.content << "\nEnter new content (or press

```

```

enter to keep current): ";
329         string newContent;
330         getline(cin, newContent);
331         if (!newContent.empty()) {
332             postToEdit.content = newContent;
333         }
334
335         cout << "Current tags: " << join(postToEdit.tags, ',') << "\nEnter new tags
(comma-separated, or press enter to keep current): ";
336         string newTags;
337         getline(cin, newTags);
338         if (!newTags.empty()) {
339             postToEdit.tags = splitString(newTags, ',');
340             for (auto& tag : postToEdit.tags) {
341                 tag = trim(tag);
342             }
343         }
344
345         postToEdit.timestamp = getCurrentTimestamp();
346
347         outFile << postToEdit.postId << "|" << postToEdit.title << "|" << postToEdit.content <<
"|"
348             << postToEdit.author << "|" << postToEdit.timestamp << "|" << join(postToEdit.
tags, ',') << endl;
349         continue;
350     }
351 }
352 outFile << line << endl;
353 }
354
355 inFile.close();
356 outFile.close();
357 remove(postFile.c_str());
358 rename("temp.txt", postFile.c_str());
359
360 if (found) {
361     cout << "Post updated successfully!\n";
362 }
363 else {
364     cout << "Post not found or you don't have permission to edit it!\n";
365 }
366 return found;
367 }
368
369 vector<Post> getUserPosts(const string &username) {
370     vector<Post> posts;
371     ifstream file(postFile);
372     string line;
373     while (getline(file, line)) {
374         vector<string> parts = splitString(line, '|');
375         if (parts.size() >= 6 && parts[3] == username) {
376             posts.push_back(parsePost(parts));
377         }
378     }
379     return posts;
380 }
381
382 vector<Post> getAllPosts() {
383     vector<Post> posts;
384     ifstream file(postFile);
385     string line;
386     while (getline(file, line)) {
387         vector<string> parts = splitString(line, '|');
388         if (parts.size() >= 6) {
389             posts.push_back(parsePost(parts));
390         }
391     }

```

```

391     }
392     return posts;
393 }
394
395 vector<Post> searchPostsByTag(const string &tag) {
396     vector<Post> results;
397     ifstream file(postFile);
398     string line;
399     while (getline(file, line)) {
400         vector<string> parts = splitString(line, '|');
401         if (parts.size() >= 6) {
402             Post post = parsePost(parts);
403             for (const auto &t : post.tags) {
404                 if (t == tag) {
405                     results.push_back(post);
406                     break;
407                 }
408             }
409         }
410     }
411     return results;
412 }
413
414 vector<Post> searchPostsByUsername(const string &username) {
415     return getUserPosts(username);
416 }
417
418 void displayPosts(const vector<Post> &posts) {
419     if (posts.empty()) {
420         cout << "No posts found.\n";
421         return;
422     }
423     for (const auto &post : posts) {
424         cout << "\nID: " << post.postId << "\nTitle: " << post.title
425             << "\nContent: " << post.content << "\nAuthor: " << post.author
426             << "\nTags: " << join(post.tags, ',') << "\nDate: " << post.timestamp << "\n";
427     }
428 }
429
430 bool deletePost(int postId, const string &currentUser, bool isAdmin) {
431     ifstream inFile(postFile);
432     ofstream outFile("temp.txt");
433     bool found = false;
434     string line;
435
436     while (getline(inFile, line)) {
437         vector<string> parts = splitString(line, '|');
438         if (!parts.empty() && stoi(parts[0]) == postId) {
439             if (parts[3] == currentUser || isAdmin) {
440                 found = true;
441                 continue;
442             }
443         }
444         outFile << line << endl;
445     }
446
447     inFile.close();
448     outFile.close();
449     remove(postFile.c_str());
450     rename("temp.txt", postFile.c_str());
451
452     // Also delete associated comments
453     if (found) {
454         deleteCommentsForPost(postId);
455     }
456     return found;

```

```

457     }
458
459     void addComment(int postId, const string &author) {
460         Comment newComment;
461         newComment.commentId = generateCommentId();
462         newComment.postId = postId;
463         newComment.author = author;
464         cout << "Enter your comment: ";
465         getline(cin, newComment.content);
466         newComment.timestamp = getCurrentTimestamp();
467
468         ofstream file(commentFile, ios::app);
469         file << newComment.commentId << "|" << newComment.postId << "|"
470             << newComment.author << "|" << newComment.content << "|"
471             << newComment.timestamp << endl;
472         cout << "Comment added successfully!\n";
473     }
474
475     vector<Comment> getCommentsForPost(int postId) {
476         vector<Comment> comments;
477         ifstream file(commentFile);
478         string line;
479         while (getline(file, line)) {
480             vector<string> parts = splitString(line, '|');
481             if (parts.size() >= 5 && stoi(parts[1]) == postId) {
482                 comments.push_back(parseComment(parts));
483             }
484         }
485         return comments;
486     }
487
488     void displayComments(const vector<Comment> &comments) {
489         if (comments.empty()) {
490             cout << "No comments yet.\n";
491             return;
492         }
493         for (const auto &comment : comments) {
494             cout << "\nID: " << comment.commentId << "\nAuthor: " << comment.author
495                 << "\nComment: " << comment.content << "\nDate: " << comment.timestamp << "\n";
496         }
497     }
498
499     bool deleteComment(int commentId, const string &currentUser, bool isAdmin) {
500         ifstream inFile(commentFile);
501         ofstream outFile("temp.txt");
502         bool found = false;
503         string line;
504
505         while (getline(inFile, line)) {
506             vector<string> parts = splitString(line, '|');
507             if (!parts.empty() && stoi(parts[0]) == commentId) {
508                 if (parts[2] == currentUser || isAdmin) {
509                     found = true;
510                     continue;
511                 }
512             }
513             outFile << line << endl;
514         }
515
516         inFile.close();
517         outFile.close();
518         remove(commentFile.c_str());
519         rename("temp.txt", commentFile.c_str());
520         return found;
521     }
522

```



```

523     bool editComment(int commentId, const string &currentUser) {
524         ifstream inFile(commentFile);
525         ofstream outFile("temp.txt");
526         bool found = false;
527         string line;
528         Comment commentToEdit;
529
530         while (getline(inFile, line)) {
531             vector<string> parts = splitString(line, '|');
532             if (!parts.empty() && stoi(parts[0]) == commentId) {
533                 if (parts[2] == currentUser) {
534                     found = true;
535                     commentToEdit = parseComment(parts);
536
537                     cout << "Current comment: " << commentToEdit.content << "\nEnter new comment: ";
538                     string newContent;
539                     getline(cin, newContent);
540                     if (!newContent.empty()) {
541                         commentToEdit.content = newContent;
542                     }
543
544                     commentToEdit.timestamp = getCurrentTimestamp();
545
546                     outFile << commentToEdit.commentId << "|" << commentToEdit.postId << "|"
547                         << commentToEdit.author << "|" << commentToEdit.content << "|"
548                         << commentToEdit.timestamp << endl;
549                     continue;
550                 }
551             }
552             outFile << line << endl;
553         }
554
555         inFile.close();
556         outFile.close();
557         remove(commentFile.c_str());
558         rename("temp.txt", commentFile.c_str());
559
560         if (found) {
561             cout << "Comment updated successfully!\n";
562         }
563         else {
564             cout << "Comment not found or you don't have permission to edit it!\n";
565         }
566         return found;
567     }
568
569 private:
570     void deleteCommentsForPost(int postId) {
571         ifstream inFile(commentFile);
572         ofstream outFile("temp.txt");
573         string line;
574
575         while (getline(inFile, line)) {
576             vector<string> parts = splitString(line, '|');
577             if (!parts.empty() && stoi(parts[1]) != postId) {
578                 outFile << line << endl;
579             }
580         }
581
582         inFile.close();
583         outFile.close();
584         remove(commentFile.c_str());
585         rename("temp.txt", commentFile.c_str());
586     }
587 };
588

```

```

589 class AdminManager {
590 private:
591     UserManager* userManager;
592     BlogManager* blogManager;
593
594     void deleteUser() {
595         string username;
596         cout << "Enter username to delete: ";
597         getline(cin, username);
598
599         if (username == userManager->getCurrentUser()) {
600             cout << "You cannot delete yourself!\n";
601             return;
602         }
603
604         ifstream inFile("users.txt");
605         ofstream outFile("temp.txt");
606         string line;
607         bool found = false;
608
609         while (getline(inFile, line)) {
610             vector<string> parts = splitString(line, '|');
611             if (!parts.empty() && parts[0] == username) {
612                 found = true;
613                 continue;
614             }
615             outFile << line << endl;
616         }
617
618         inFile.close();
619         outFile.close();
620         remove("users.txt");
621         rename("temp.txt", "users.txt");
622
623         if (found) {
624             cout << "User deleted successfully!\n";
625             // Also delete user's posts and comments
626             vector<Post> userPosts = blogManager->getUserPosts(username);
627             for (const auto& post : userPosts) {
628                 blogManager->deletePost(post.postId, username, true);
629             }
630         }
631         else {
632             cout << "User not found!\n";
633         }
634     }
635
636     void makeAdmin() {
637         string username;
638         cout << "Enter username to make admin: ";
639         getline(cin, username);
640
641         ifstream inFile("users.txt");
642         ofstream outFile("temp.txt");
643         string line;
644         bool found = false;
645
646         while (getline(inFile, line)) {
647             vector<string> parts = splitString(line, '|');
648             if (!parts.empty() && parts[0] == username) {
649                 found = true;
650                 parts[3] = "1";
651                 outFile << join(parts, '|') << endl;
652             }
653             else {
654                 outFile << line << endl;

```

```

655         }
656     }
657
658     inFile.close();
659     outFile.close();
660     remove("users.txt");
661     rename("temp.txt", "users.txt");
662
663     if (found) {
664         cout << "User is now an admin!\n";
665     }
666     else {
667         cout << "User not found!\n";
668     }
669 }
670
671 void deleteAnyPost() {
672     int postId;
673     cout << "Enter post ID to delete: ";
674     cin >> postId;
675     clearInputBuffer();
676
677     if (blogManager->deletePost(postId, "", true)) {
678         cout << "Post deleted successfully!\n";
679     }
680     else {
681         cout << "Post not found!\n";
682     }
683 }
684
685 public:
686     AdminManager(UserManager* um, BlogManager* bm) : userManager(um), blogManager(bm) {}
687
688     void adminMenu() {
689         while (true) {
690             cout << "\nAdmin Panel\n1. Delete User\n2. Make User Admin\n3. Delete Any Post\n4.
Back\nChoice: ";
691             int choice;
692             cin >> choice;
693             clearInputBuffer();
694
695             switch (choice) {
696                 case 1:
697                     deleteUser();
698                     break;
699                 case 2:
700                     makeAdmin();
701                     break;
702                 case 3:
703                     deleteAnyPost();
704                     break;
705                 case 4:
706                     return;
707                 default:
708                     cout << "Invalid choice!\n";
709             }
710         }
711     }
712 };
713
714 class TagManager {
715 private:
716     BlogManager* blogManager;
717
718 public:
719     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 ()) {
766                     targetPost.tags.push_back(newTag);
767                     blogManager->editPost(postId, currentUser); // This will trigger a reload
768                     cout << "Tag added successfully!\n";
769                 }
770                 else {
771                     cout << "Tag already exists!\n";
772                 }
773             }
774         }
775         else if (choice == 2) {
776             cout << "Enter tag to delete: ";
777             string tagToDelete;
778             getline(cin, tagToDelete);
779             tagToDelete = trim(tagToDelete);
780
781             auto it = find(targetPost.tags.begin(), targetPost.tags.end(), tagToDelete);
782             if (it != targetPost.tags.end()) {
783                 targetPost.tags.erase(it);
784                 blogManager->editPost(postId, currentUser); // This will trigger a reload
785                 cout << "Tag deleted successfully!\n";

```

```

785         }
786         else {
787             cout << "Tag not found!\n";
788         }
789     }
790     else if (choice == 3) {
791         break;
792     }
793     else {
794         cout << "Invalid choice!\n";
795     }
796 }
797 };
798 };
799
800 // Main application
801 int main() {
802     UserManager userManager;
803     BlogManager blogManager;
804     AdminManager adminManager(&userManager, &blogManager);
805     TagManager tagManager(&blogManager);
806
807     cout<<"Welcome to USER BLOG MANAGEMENT SYSTEM!"<<endl;
808
809     while (true) {
810         if (!userManager.isLoggedIn()) {
811             cout << "\nMain Menu\n1. Register\n2. Login\n3. Exit\nChoice: ";
812             int choice;
813             cin >> choice;
814             clearInputBuffer();
815
816             if (choice == 1) {
817                 userManager.registerUser();
818             }
819             else if (choice == 2) {
820                 userManager.login();
821             }
822             else if (choice == 3) {
823                 break;
824             }
825             else {
826                 cout << "Invalid choice!\n";
827             }
828         }
829         else {
830             cout << "\nWelcome " << userManager.getCurrentUser() << "!\n";
831             if (userManager.isUserAdmin()) {
832                 cout << "1. Admin Panel\n";
833             }
834             cout << "1. Create Post\n2. My Posts\n3. Edit Post\n4. Delete Post\n5. Manage Tags\n"
835                 << "6. Search Posts by Tag\n7. Search Posts by Username\n8. View All Posts\n"
836                 << "9. Add Comment\n10. View/Edit/Delete Comments\n11. Logout\nChoice: ";
837
838             int choice;
839             cin >> choice;
840             clearInputBuffer();
841
842             if (userManager.isUserAdmin() && choice == 1) {
843                 adminManager.adminMenu();
844             }
845             else {
846                 switch (choice) {
847                     case 1:
848                         blogManager.createPost(userManager.getCurrentUser());
849                         break;
850                     case 2: {

```

```

851         vector<Post> posts = blogManager.getUserPosts(userManager.getCurrentUser());
852         blogManager.displayPosts(posts);
853         break;
854     }
855     case 3: {
856         int postId;
857         cout << "Enter post ID to edit: ";
858         cin >> postId;
859         clearInputBuffer();
860         blogManager.editPost(postId, userManager.getCurrentUser());
861         break;
862     }
863     case 4: {
864         int postId;
865         cout << "Enter post ID to delete: ";
866         cin >> postId;
867         clearInputBuffer();
868         if (blogManager.deletePost(postId, userManager.getCurrentUser(), false)) {
869             cout << "Post deleted!\n";
870         }
871         else {
872             cout << "Post not found or unauthorized!\n";
873         }
874         break;
875     }
876     case 5:
877         tagManager.manageTags(userManager.getCurrentUser());
878         break;
879     case 6: {
880         string tag;
881         cout << "Enter tag to search: ";
882         getline(cin, tag);
883         vector<Post> results = blogManager.searchPostsByTag(tag);
884         blogManager.displayPosts(results);
885         break;
886     }
887     case 7: {
888         string username;
889         cout << "Enter username to search: ";
890         getline(cin, username);
891         vector<Post> results = blogManager.searchPostsByUsername(username);
892         blogManager.displayPosts(results);
893         break;
894     }
895     case 8: {
896         vector<Post> allPosts = blogManager.getAllPosts();
897         blogManager.displayPosts(allPosts);
898         break;
899     }
900     case 9: {
901         int postId;
902         cout << "Enter post ID to comment on: ";
903         cin >> postId;
904         clearInputBuffer();
905         blogManager.addComment(postId, userManager.getCurrentUser());
906         break;
907     }
908     case 10: {
909         int postId;
910         cout << "Enter post ID to view comments: ";
911         cin >> postId;
912         clearInputBuffer();
913
914         vector<Comment> comments = blogManager.getCommentsForPost(postId);
915         blogManager.displayComments(comments);
916

```

```

917         if (!comments.empty()) {
918             cout << "\n1. Edit Comment\n2. Delete Comment\n3. Back\nChoice: ";
919             int commentChoice;
920             cin >> commentChoice;
921             clearInputBuffer();
922
923             if (commentChoice == 1 || commentChoice == 2) {
924                 int commentId;
925                 cout << "Enter comment ID: ";
926                 cin >> commentId;
927                 clearInputBuffer();
928
929                 if (commentChoice == 1) {
930                     blogManager.editComment(commentId, userManager.getCurrentUser());
931                 }
932                 else {
933                     if (blogManager.deleteComment(commentId, userManager.getCurrentUser(),
false)) {
934                         cout << "Comment deleted!\n";
935                     }
936                     else {
937                         cout << "Comment not found or unauthorized!\n";
938                     }
939                 }
940             }
941         }
942         break;
943     }
944     case 11:
945         userManager.logout();
946         break;
947     default:
948         cout << "Invalid choice!\n";
949     }
950 }
951 }
952 }
953 return 0;
954

```