RIPHAH INTERNATIONAL UNIVERSITY, GG CAMPUS

Data Structure and Algorithms
Fall 2023

Mini Blog Management System

Project Team

Name of Students	Sap ID	Program	Valid Email Address
Nagarash Fateh	44815	BSSE	44815@students.riphah.edu.pk
Manahil Habib	47876	BSSE	47876@students.riphah.edu.pk
Sana Arshad	46189	BSSE	46189@students.riphah.edu.pk
Iman Arshad	46188	BSSE	46188@students.riphah.edu.pk
Leena Siddiqa	46963	BSSE	46963@students.riphah.edu.pk
Laiba Jameel	45943	BSSE	45943@students.riphah.edu.pk

Date of Submission
11/23/2023

Table of Contents

Artifact # 1
Project Proposal

Introduction of the Project

Project Title: Mini Blog Management System

Introduction:

The Mini Blog Management System is a comprehensive software project designed to provide users with a platform for creating, managing, and interacting with blogs. In today's digital age, blogging has become an essential means of expressing ideas, sharing information, and fostering online communities. This system aims to simplify the process of blog creation, user management, and content interaction.

Problem Statement:

In the digital age, there is a growing need for efficient and user-friendly platforms for managing and organizing blogs. Individuals and organizations often find it challenging to maintain a systematic record of users, categories, posts, comments, and feedbacks.

Proposed Solution:

the proposed Mini Blog Management System aims to provide a streamlined solution for creating, managing, and interacting with a simplified blogging environment.

Scope of the Project:

The Mini Blog Management System is designed to provide a simple and efficient platform for managing and organizing blog-related activities.

Modules Description:

1. **User Module:**

The User Module is responsible for managing user-related functionalities within the Mini Blog Management System. It includes features such as user registration, authentication, and profile management. Users can create accounts, log in securely, edit their profiles, and maintain a personalized presence on the platform.

2. **Blog Post Module:**

The Blog Post Module facilitates the creation, modification, and deletion of blog posts. Users can compose and publish content, assign categories, and edit their posts as needed.

3. **Comment Module:**

The Comment Module enables user interaction by allowing them to post comments on blog entries. Users can express their opinions, engage in discussions, and provide feedback on

the content. It includes features such as comment submission, editing, and deletion, while also maintaining a record of user interactions.

4. **Category Module:**

The Category Module focuses on the organization and categorization of blog content. Users can create, edit, and delete categories to effectively group related posts. Each blog post can be associated with one or more categories, contributing to a well-organized and user-friendly blog structure.

5. **Search Module:**

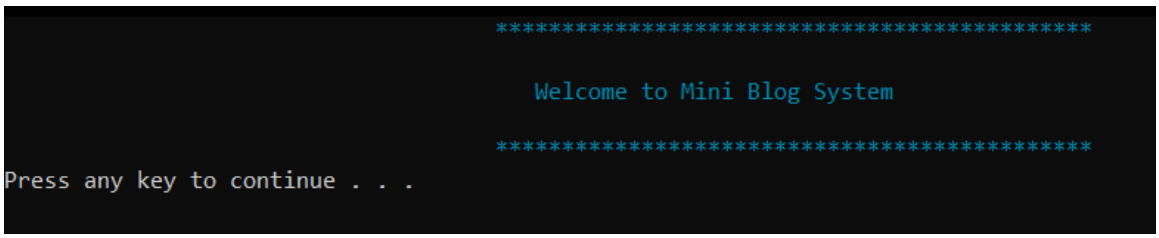
The Search Module empowers users to efficiently locate specific blog posts or topics of interest within the system. It provides a user-friendly search interface, enabling users to enter keywords, tags, or phrases to retrieve relevant content.

6. **Feedback Module:**

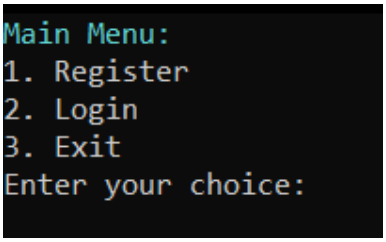
The Feedback Module captures and manages user feedback on the Mini Blog Management System. Users can submit feedback, suggestions, or report issues through a dedicated interface. The Feedback Module contributes to the continuous enhancement and refinement of the blogging platform.

Artifact # 2
Screens

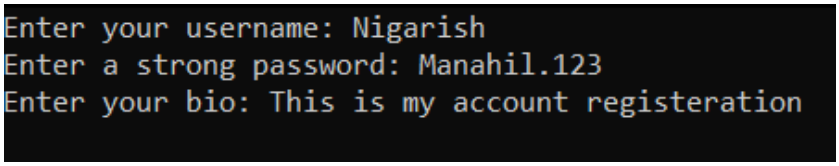
Main Banner



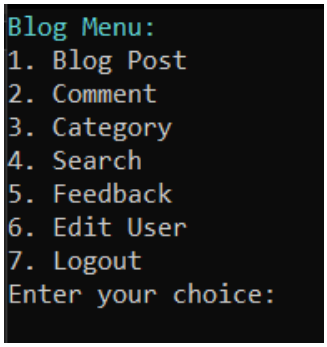
Registration Menu



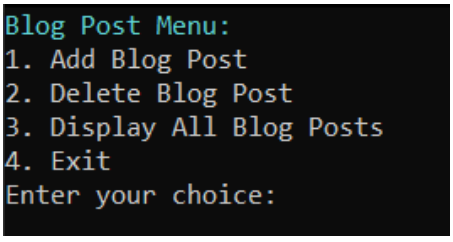
Registering

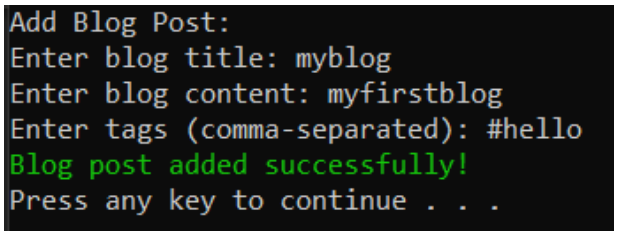


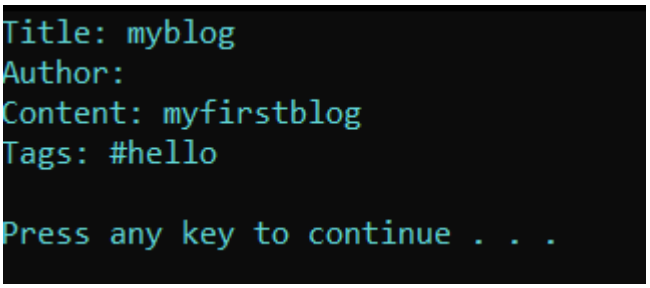
Blog's main dashboard

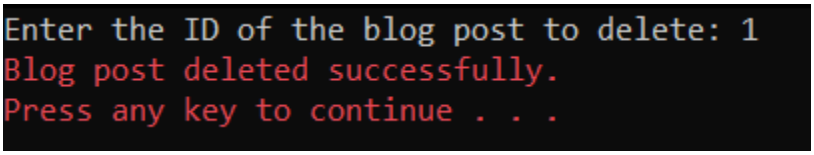


Option 1: Blog Post

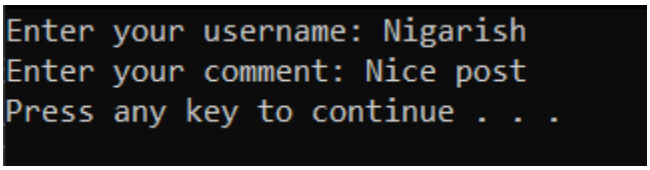


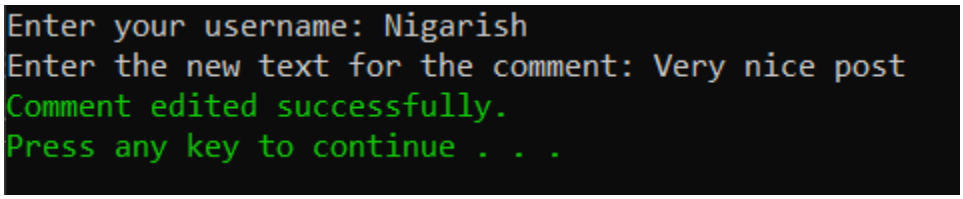


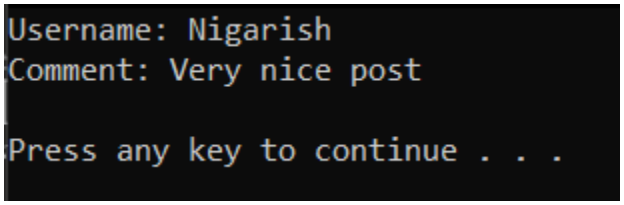




Option 2: Comment







```
Enter your username: Nigarish
Comment deleted successfully.
Press any key to continue . . .
```

Option 3: Category

```
Enter the name of category: A
Press any key to continue . . .
```

```
Enter ID of category to update: 1
Enter new name for the category: B
Category with ID 1 updated.
Press any key to continue . . .
```

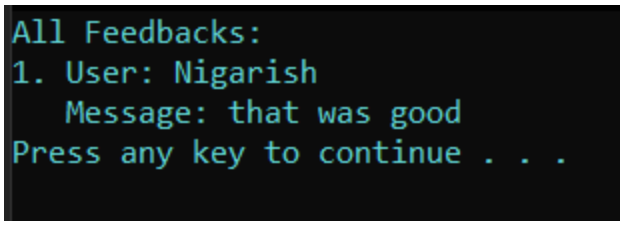
```
All Categories:
4. D
3. C
2. A
1. B
Press any key to continue . . .
```

Option 4: Search

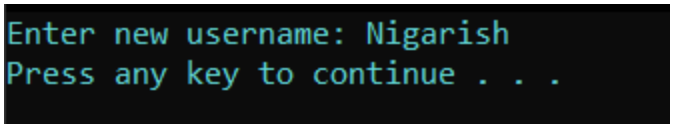
```
Press any key to continue . . .
Title: Blogpost
Content: myfirstblogpost
Tags: #hello

Press any key to continue . . .
```

Option 5: Feedback

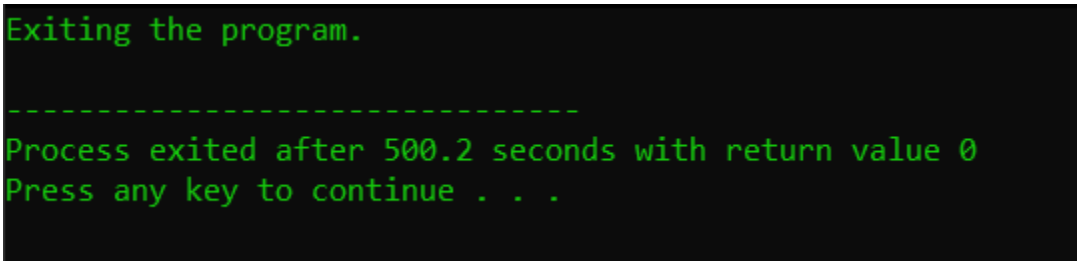
```
Enter the username: Nigarish
Enter the feedback message: that was good
Enter feedback id:1
Press any key to continue . . .
```



Option 6: Edit User



Exiting program



Artifact # 3
Code

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <string>

#ifdef _WIN32
#include <windows.h>
#else
#include <unistd.h>
#endif
#define RESET   "\033[0m"
#define COBALT_BLUE "\033[38;5;31m" // Cobalt blue color

using namespace std;

#ifdef _WIN32
const string CLEAR_SCREEN_COMMAND = "cls";
#else
const string CLEAR_SCREEN_COMMAND = "clear";
#endif

#include <iostream>
// Function to get the width of the console
int getConsoleWidth() {
#ifdef _WIN32
    // For Windows
    CONSOLE_SCREEN_BUFFER_INFO csbi;
    GetConsoleScreenBufferInfo(GetStdHandle(STD_OUTPUT_HANDLE), &csbi);
    return csbi.srWindow.Right - csbi.srWindow.Left + 1;
#else
    // For Unix-like systems
    struct winsize size;
    ioctl(STDOUT_FILENO, TIOCGWINSZ, &size);
    return size.ws_col;
#endif
}

void displayBanner()  {
   int consoleWidth = getConsoleWidth();
    int bannerWidth = 45; // Width of the banner text

// Calculate left padding to center the text
    int padding = (consoleWidth - bannerWidth) / 2;
```

```
// Display the centered colored banner
    cout << string(padding, ' ') << COBALT_BLUE <<
"*******************************************" << RESET << endl;
    cout << string(padding, ' ') << COBALT_BLUE << "                                         "
<< RESET << endl;
    cout << string(padding, ' ') << COBALT_BLUE << "   " <<       "Welcome to Mini
Blog System" << " " << RESET << endl;
    cout << string(padding, ' ') << COBALT_BLUE << "                                         "
<< RESET << endl;
    cout << string(padding, ' ') << COBALT_BLUE <<
"*******************************************" << RESET << endl;
}

// Function to clear the console screen
void clearScreen() {
    system(CLEAR_SCREEN_COMMAND.c_str());
}

// Function to add color to console output (for Windows only)
#ifdef _WIN32
ostream& blue(ostream& s) {
    SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE),
FOREGROUND_BLUE | FOREGROUND_GREEN |
FOREGROUND_INTENSITY);
    return s;
}

ostream& red(ostream& s) {
    SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE),
FOREGROUND_RED | FOREGROUND_INTENSITY);
    return s;
}

ostream& green(ostream& s) {
    SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE),
FOREGROUND_GREEN | FOREGROUND_INTENSITY);
    return s;
}

ostream& white(ostream& s) {
    SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE),
FOREGROUND_RED | FOREGROUND_GREEN | FOREGROUND_BLUE);
    return s;
}
```

```
#else
ostream& blue(ostream& s) {
    return s << "\033[1;34m";
}

ostream& red(ostream& s) {
    return s << "\033[1;31m";
}

ostream& green(ostream& s) {
    return s << "\033[1;32m";
}

ostream& white(ostream& s) {
    return s << "\033[0m";
}
#endif

// User structure
struct User {
    string username;
    string password;
    string bio;
    User* next;
};


// Function to check if a password is strong
bool isStrongPassword(const string& password) {
    bool hasUppercase = false;
    bool hasLowercase = false;
    bool hasDigit = false;
    bool hasSymbol = false;

    for (size_t i = 0; i < password.length(); ++i) {
        char ch = password[i];

        if (isupper(ch)) {
            hasUppercase = true;
        } else if (islower(ch)) {
            hasLowercase = true;
        } else if (isdigit(ch)) {
            hasDigit = true;
        } else if (ispunct(ch)) {
            hasSymbol = true;
        }
```

```
if (!(hasUppercase && hasLowercase && hasDigit && hasSymbol)) {
        cout <<red<< "Password must contain a combination of uppercase letters,
lowercase letters, numbers, and symbols.\n"<<white;
        return false;
    }

    return true;
}

// Function to register a new user
void registerUser(User*& userList) {
            system("cls");
    User* newUser = new User;

    cout << "Enter your username: ";
    cin >> newUser->username;

    do {
        cout << "Enter a strong password: ";
        cin >> newUser->password;
    } while (!isStrongPassword(newUser->password));

    cout << "Enter your bio: ";
    cin.ignore();
    getline(cin, newUser->bio);

    newUser->next = userList;
    userList = newUser;

    ofstream outputFile("user_data.txt", ios::app);
    if (outputFile.is_open()) {
        outputFile << newUser->username << " " << newUser->password << " " <<
newUser->bio << endl;
        outputFile.close();
    } else {
        cerr <<red<< "Unable to open file for writing." << endl;
    }
}

// Function to authenticate a user
bool authenticateUser(const string& username, const string& password, User*
userList) {
    User* current = userList;
    while (current != NULL) {
```

```
if (current->username == username && current->password == password) {
            cout <<green<< "Login successful!" << endl;
            return true;
        }
        current = current->next;
    }

    cout <<red<< "Invalid username or password." << endl;
    return false;
}

// Function to edit user information
void editUser(User*& userList, const string& username) {
    User* current = userList;
    while (current != NULL) {
        if (current->username == username) {
            cout <<blue<< "Edit Options:\n";
            cout << "1. Edit Username\n";
            cout << "2. Edit Password\n";
            cout << "3. Edit Bio\n";
            cout << "4. Back to Main Menu\n";
            cout << "Enter your choice: ";
            int editChoice;
            cin >> editChoice;

            switch (editChoice) {
                case 1:
                    clearScreen();
                    cout <<"Enter new username: ";
                    cin >> current->username;
                    system("pause");
                    break;
                case 2:
                    clearScreen();
                    do {
                        cout << "Enter a strong password: ";
                        cin >> current->password;
                    } while (!isStrongPassword(current->password));
                    system("pause");
                    break;
                case 3:
                    clearScreen();
                    cout <<"Enter new bio: ";
                    cin.ignore();
```

```
getline(cin, current->bio);
                system("pause");
                break;
            case 4:{
                cout<<green<<"Exiting...\n";
                                        break;
                                    }
            default:
                cout <<red<< "Invalid choice. Please try again.\n";
        }

        ofstream outputFile("user_data.txt");
        User* temp = userList;
        while (temp != NULL) {
            outputFile << temp->username << " " << temp->password << " " << temp-
>bio << endl;
            temp = temp->next;
        }
        outputFile.close();
    }
    current = current->next;
  }
}

// Function to load user data from a file
void loadUserDataFromFile(User*& userList) {
        system("cls");
  ifstream inputFile("user_data.txt");


  if (inputFile.is_open()) {
    User* newUser;
    while (!inputFile.eof()) {
      newUser = new User;
      inputFile >> newUser->username >> newUser->password;
      getline(inputFile, newUser->bio);
      newUser->next = userList;
      userList = newUser;
    }
    inputFile.close();
  }
}
```

```
// Struct for a single category
struct CategoryNode {
    int cid;
    string name;
    CategoryNode* next;

    CategoryNode(int id, const string& categoryName)
        : cid(id), name(categoryName), next(NULL) {}
};

class CategoryLinkedList {
private:
    CategoryNode* head;
    int size; // Added variable to track the size

public:
    CategoryLinkedList() : head(NULL), size(0) {}

    void addCategory(int id, const string& categoryName) {
        CategoryNode* newNode = new CategoryNode(id, categoryName);
        newNode->next = head;
        head = newNode;
        size++;
    }

    CategoryNode* findCategory(int id) {
        CategoryNode* current = head;
        while (current != NULL && current->cid != id) {
            current = current->next;
        }
        return current;
    }

void deleteCategory(int id) {
        CategoryNode* current = head;
        CategoryNode* prev = NULL;

        while (current != NULL && current->cid != id) {
            prev = current;
            current = current->next;
        }
```

```
if (current == NULL) {
          cout << "Category with ID " << id <<red<< " not found." <<endl;
          return;
       }

       if (prev == NULL) {
          head = current->next;
       } else {
          prev->next = current->next;
       }

       delete current;
       size--;
       cout << "Category with ID " << id <<red<< " deleted." <<endl;
    }

    void updateCategory(int id, const string& newName) {
       CategoryNode* foundCategory = findCategory(id);
       if (foundCategory != NULL) {
          foundCategory->name = newName;
          cout << "Category with ID " << id <<green<< " updated." <<endl;
       } else {
          cout << "Category with ID " << id <<red<< " not found." << endl;
       }
    }

    void displayCategories() const {
       CategoryNode* current = head;
       while (current != NULL) {
          cout << current->cid << ". " << current->name << endl;
          current = current->next;
       }
    }

    int getSize() const {
       return size;
    }

    ~CategoryLinkedList() {
       while (head != NULL) {
          CategoryNode* temp = head;
          head = head->next;
          delete temp;
       }
    }
};
```

```
class CategoryManager {
private:
    CategoryLinkedList categories;

public:
    void addCategory() {
        string categoryName;
        cout << "Enter the name of category: ";
        cin >> categoryName;
        int id = categories.getSize() + 1;
        categories.addCategory(id, categoryName);
    }

    void deleteCategory() {
        int cid;
        cout <<"Enter ID of category to delete: ";
        cin >> cid;
        categories.deleteCategory(cid);
    }

    void updateCategory() {
        int cid;
        string newName;
        cout << "Enter ID of category to update: ";
        cin >> cid;
        cout << "Enter new name for the category: ";
        cin >> newName;
        categories.updateCategory(cid, newName);
    }

    void displayCategories() {
        cout <<blue<< "All Categories:" << endl;
        categories.displayCategories();
    }

};

// Comment structure
struct Comment {
    string username;
    string text;
    Comment* next;

    Comment(const string& u, const string& t, Comment* n = NULL)
        : username(u), text(t), next(n) {}
};
```

```
// Class for managing comments
class CommentList {
private:
    Comment* head;
public:
    CommentList() : head(NULL) {}

    void editComment(const string& username) {
        Comment* current = head;
        while (current != NULL) {
            if (current->username == username) {
                cout << "Enter the new text for the comment: ";
                getline(cin, current->text);
                cout <<green<< "Comment edited successfully.\n";
                return;
            }
            current = current->next;
        }
        cout <<red<< "Comment not found.\n";
    }

    void deleteComment(const string& username) {
        Comment* current = head;
        Comment* prev = NULL;

        while (current != NULL) {
            if (current->username == username) {
                if (prev != NULL) {
                    prev->next = current->next;
                } else {
                    head = current->next;
                }

                delete current;
                cout <<red<< "Comment deleted successfully.\n";
                return;
            }

            prev = current;
            current = current->next;
        }

        cout <<red<< "Comment not found.\n";
    }
```

```
  void addComment(const string& username, const string& text) {
      Comment* newComment = new Comment(username, text, head);
      head = newComment;
   }

   void displayComments() const {
      Comment* current = head;

      while (current != NULL) {
         cout << "Username: " << current->username << endl;
         cout << "Comment: " << current->text << endl << endl;
         current = current->next;
      }
   }
};

// Struct for a single feedback
struct FeedbackNode {
   int fid;
   string user;
   string message;
   FeedbackNode* next;

   FeedbackNode(int id, const string& userName, const string& text)
      : fid(id), user(userName), message(text), next(NULL) {}
};

class FeedbackLinkedList {
private:
   FeedbackNode* head;
   int size; // Added variable to track the size

public:
   FeedbackLinkedList() : head(NULL), size(0) {}

   void addFeedback(int id, const string& userName, const string& text) {
      FeedbackNode* newNode = new FeedbackNode(id, userName, text);
      newNode->next = head;
      head = newNode;
      size++;
   }

   FeedbackNode* findFeedback(int id) {
      FeedbackNode* current = head;
      while (current != NULL && current->fid != id) {
```

```
current = current->next;
        }
        return current;
    }

    void deleteFeedback(int id) {
        FeedbackNode* current = head;
        FeedbackNode* prev = NULL;

        while (current != NULL && current->fid != id) {
            prev = current;
            current = current->next;
        }

        if (current == NULL) {
            cout << "Feedback with ID " << id <<red<< " not found." << endl;
            return;
        }

        if (prev == NULL) {
            head = current->next;
        } else {
            prev->next = current->next;
        }

        delete current;
        size--;
        cout << "Feedback with ID " << id <<red<< " deleted." << endl;
    }

    void displayFeedbacks() const {
        FeedbackNode* current = head;
        while (current != NULL) {
            cout << current->fid << ". User: " << current->user << "\n   Message: " <<
current->message << endl;
            current = current->next;
        }
    }

int getSize() const {
        return size;
    }
```

```
~FeedbackLinkedList() {
        while (head != NULL) {
            FeedbackNode* temp = head;
            head = head->next;
            delete temp;
        }
    }
};

class FeedbackManager {
private:
    FeedbackLinkedList feedbacks;
    int fid;
public:
    void addFeedback() {
        string userName, message;
        cout << "Enter the username: ";
        cin >> userName;
        cout << "Enter the feedback message: ";
        cin.ignore();
        getline(std::cin, message);
        int id = feedbacks.getSize() + 1;
        cout<<blue<<"Enter feedback id:";
        cin>>fid;
        feedbacks.addFeedback(id, userName, message);
    }

    void deleteFeedback() {
        int fid;
        cout << "Enter ID of feedback to delete: ";
        cin >> fid;
        feedbacks.deleteFeedback(fid);
    }

    void displayFeedbacks() {
        cout <<blue<< "All Feedbacks:" << std::endl;
        feedbacks.displayFeedbacks();
    }

};
```

```
// BlogPost structure
struct BlogPost {
    string title;
    string author;
    string content;
    string tags;
    CommentList comments;  // Include CommentList to manage comments
    BlogPost* next;

    BlogPost(const string& t, const string& a, const string& c, const string& tg)
        : title(t), author(a), content(c), tags(tg), next(NULL) {}
};

// Function to add a blog post to the linked list
void addBlogPost(BlogPost*& head, const string& title, const string& author, const
string& content, const string& tags) {
            BlogPost* newPost = new BlogPost(title, author, content, tags);
    newPost->next = head; // Insert at the beginning
    head = newPost;
}

// Function to search for blog posts
void searchBlog(const BlogPost* head, int choice, const string& keyword) {
    const BlogPost* current = head;
    bool found = false;

    while (current != NULL) {
        bool match = false;
        switch (choice) {
            case 1:
                    clearScreen();
                match = (current->title.find(keyword) != string::npos);
                system("pause");
                break;
            case 2:
                    clearScreen();
                match = (current->author.find(keyword) != string::npos);
                system("pause");
                break;
            case 3:
                    clearScreen();
                match = (current->tags.find(keyword) != string::npos);
                system("pause");
```

```
break;
            default:
                cout <<red<< "Invalid search choice.\n";
                return;
        }

        if (match) {
            found = true;
            cout <<blue<< "Title: " << current->title << endl;
            cout <<blue<< "Content: " << current->content << endl;
            cout <<blue<< "Tags: " << current->tags << endl << endl;
        }
        current = current->next;
    }

    if (!found) {
        cout <<red<< "No matching blog posts found.\n";
    }
}

// Function to delete a blog post
void deleteBlogPost(BlogPost*& head, int id) {
    BlogPost* current = head;
    BlogPost* previous = NULL;
    int currentId = 1;

    while (current != NULL && currentId != id) {
        previous = current;
        current = current->next;
        currentId++;
    }

    if (current == NULL) {
        cout <<red<< "Invalid blog post ID.\n";
        return;
    }

    if (previous == NULL) {
        head = current->next;
    } else {
        previous->next = current->next;
    }

    delete current;
    cout <<red<< "Blog post deleted successfully.\n";
}
```

```
// Function to display all blog posts
void displayBlogPosts(const BlogPost* head) {
    const BlogPost* current = head;

     while (current != NULL) {
        cout <<blue<< "Title: " << current->title << endl;
        cout <<blue<< "Author: " << current->author << endl;
        cout <<blue<< "Content: " << current->content << endl;
        cout <<blue<< "Tags: " << current->tags << endl << endl;
        current = current->next;
    }
}

int main() {
    // Initialize linked lists for users and blog posts
    User* userList = NULL;
    BlogPost* blogPostList = NULL;

    // Load user data from a file
    loadUserDataFromFile(userList);

            // Initialize CommentList, CategoryManager, and FeedbackManager
            CommentList commentList;
            CategoryManager categoryManager;
            FeedbackManager feedbackManager;

    int choice;

    // Display the program banner
    displayBanner();
    system("pause");

    while (true) {
        // Clear the console screen for the main menu
        clearScreen();

        // Display the main menu
        cout << blue << "Main Menu:" << white << endl;
        cout << "1. Register\n";
        cout << "2. Login\n";
        cout << "3. Exit\n";
        cout << "Enter your choice: ";
        int mainChoice;
        cin >> mainChoice;
```

```
switch (mainChoice) {
        case 1:{
            // Register a new user
            clearScreen();
            cout <<  "Register a new user:" << white << endl;
            registerUser(userList);
            break;
        }
        case 2:{
            // Login
            string username, password;
            clearScreen();
            cout << blue << "Login:" << white << endl;
            cout << "Enter your username: ";
            cin >> username;
            cout << "Enter your password: ";
            cin >> password;
            authenticateUser(username, password, userList);
            break;
        }
        case 3:{
            // Exit the program
            clearScreen();
            cout <<green<< "Exiting the program.\n";
            return 0;
        }
        default:
            cout << red << "Invalid choice. Please try again.\n" << white;
    }

    // If there is a logged-in user
    if (userList != NULL) {
        string currentUser = userList->username;

        while (true) {
            // Clear the console screen for the blog menu
            clearScreen();

            // Display the blog menu
            cout << blue << "Blog Menu:" << white << endl;
```

```
cout << "1. Blog Post\n";
            cout << "2. Comment\n";
            cout << "3. Category\n";
            cout << "4. Search\n";
            cout << "5. Feedback\n";
            cout << "6. Edit User\n";
            cout << "7. Logout\n";
            cout << "Enter your choice: ";
            int blogChoice;
            cin >> blogChoice;

            switch (blogChoice) {
               case 1:{
                  // Blog Post Menu
                  while (true) {
                     clearScreen();
                     cout << blue << "Blog Post Menu:" << white << endl;
                     cout << "1. Add Blog Post\n";
                     cout << "2. Delete Blog Post\n";
                     cout << "3. Display All Blog Posts\n";
                     cout << "4. Exit\n";
                     cout << "Enter your choice: ";
                     int blogChoice;
                     cin >> blogChoice;

                     switch (blogChoice) {
                        case 1: {
                           // Add Blog Post
                           clearScreen();
                           cout << "Add Blog Post:" << white << endl;
                           string title, content, tags;
                           cout << "Enter blog title: ";
                           cin.ignore();
                           getline(cin, title);
                           cout << "Enter blog content: ";
                           getline(cin, content);
                           cout << "Enter tags (comma-separated): ";
                           getline(cin, tags);
                           addBlogPost(blogPostList, title, currentUser, content, tags);
                           cout << green << "Blog post added successfully!\n" << white;
                           system("pause");
                           break;
                        }
                        case 2: {
                           // Delete Blog Post
                           clearScreen();
```

```
cout << "Enter the ID of the blog post to delete: ";
                    cin >> deleteId;
                    deleteBlogPost(blogPostList, deleteId);
                    system("pause");
                    break;
                }
                case 3: {
                    // Display All Blog Posts
                    clearScreen();
                    displayBlogPosts(blogPostList);
                    system("pause");
                    break;
                }
                case 4: {
                    // Exit the Blog Post Menu
                    cout << green << "Exiting...\n";
                    break;
                }
                default: {
                    cout << red << "Invalid choice. Please try again.\n" << white;
                    system("pause");
                }
            }
            // Exit the Blog Post Menu if the user chooses to log out
            if (blogChoice == 4) {
                break;
            }
        }
        break;
    }

    case 2: {
        // Comment Menu
        while (true) {
            clearScreen();
            cout << blue << "Comment Menu:" << white << endl;
            cout << "1. Add Comment\n";
            cout << "2. Edit Comment\n";
            cout << "3. Delete Comment\n";
            cout << "4. Display Comments\n";
            cout << "5. Exit\n";
            cout << "Choose an option: ";

            int choice;
            cin >> choice;
```

```
cin.ignore();

                switch (choice) {
                    case 1: {
                        // Add Comment
                        clearScreen();
                        string username, text;
                        cout << "Enter your username: ";
                        getline(cin, username);
                        cout << "Enter your comment: ";
                        getline(cin, text);
                        commentList.addComment(username, text);
                        system("pause");
                        break;
                    }
                    case 2: {
                        // Edit Comment
                        clearScreen();
                        string username;
                        cout << "Enter your username: ";
                        getline(cin, username);
                        commentList.editComment(username);
                        system("pause");
                        break;
                    }
                    case 3: {
                        // Delete Comment
                        clearScreen();
                        string username;
                        cout << "Enter your username: ";
                        getline(cin, username);
                        commentList.deleteComment(username);
                        system("pause");
                        break;
                    }
                    case 4: {
                        // Display Comments
                        clearScreen();
                        commentList.displayComments();
                        system("pause");
                        break;
                    }
                    case 5: {
                        // Exit the Comment Menu
                        cout << green << "Exiting...\n";
                        break;
```

```
}
                    default: {
                        cout << red << "Invalid choice. Try again.\n";
                    }
                }
                // Exit the Comment Menu if the user chooses to log out
                if (choice == 5) {
                    break;
                }
            }
            break;
        }

        case 3: {
            // Category Menu
            do {
                clearScreen();
                cout << blue << "Category Menu:" << white << endl;
                cout <<
                    "1. Add Category\n"
                    "2. Delete Category\n"
                    "3. Update Category\n"
                    "4. Display Categories\n"
                    "5. Exit\n"
                    "Enter your choice: ";
                cin >> choice;

                switch (choice) {
                    case 1: {
                        // Add Category
                        clearScreen();
                        categoryManager.addCategory();
                        system("pause");
                        break;
                    }
                    case 2: {
                        // Delete Category
                        clearScreen();
                        categoryManager.deleteCategory();
                        system("pause");
                        break;
                    }
                    case 3: {
                        // Update Category
                        clearScreen();
```

```
categoryManager.updateCategory();
                        system("pause");
                        break;
                    }
                    case 4: {
                        // Display Categories
                        clearScreen();
                        categoryManager.displayCategories();
                        system("pause");
                        break;
                    }
                    case 5: {
                        // Exit the Category Menu
                        cout << "Exiting...\n";
                        break;
                    }
                    default: {
                        cout << red << "Invalid choice. Please enter a valid option." <<
endl;
                    }
                }
            } while (choice != 5);
            break;
        }

        case 4: {
            // Search Menu
            clearScreen();
            int searchChoice;
            cout << blue << "Search Menu:" << white << endl;
            cout << blue << "Search by:\n";
            cout << "1. Title\n";
            cout << "2. Author\n";
            cout << "3. Tags\n";
            cout << "Enter your choice: ";
            cin >> searchChoice;
            cout << "Enter search keyword: ";
            string keyword;
            cin.ignore();
            getline(cin, keyword);
            searchBlog(blogPostList, searchChoice, keyword);
            system("pause");
            break;
        }
```

```
case 5: {
                    // Feedback Menu
                    do {
                        clearScreen();
                        cout << blue << "Feedback Menu:" << white << endl;
                        cout <<
                            "1. Add Feedback\n"
                            "2. Delete Feedback\n"
                            "3. Display Feedbacks\n"
                            "4. Exit\n"
                            "Enter your choice: ";
                        cin >> choice;

                        switch (choice) {
                            case 1: {
                                // Add Feedback
                                clearScreen();
                                feedbackManager.addFeedback();
                                system("pause");
                                break;
                            }
                            case 2: {
                                // Delete Feedback
                                clearScreen();
                                feedbackManager.deleteFeedback();
                                system("pause");
                                break;
                            }
                            case 3: {
                                // Display Feedbacks
                                clearScreen();
                                feedbackManager.displayFeedbacks();
                                system("pause");
                                break;
                            }
                            case 4: {
                                // Exit the Feedback Menu
                                cout << green << "Exiting...\n";
                                break;
                            }
                            default: {
                                cout << red << "Invalid choice. Please enter a valid option.\n";
                            }
                        }
                    } while (choice != 4);
                    Break;
```

```
}

                case 6: {
                    // Edit User
                    clearScreen();
                    editUser(userList, currentUser);
                    break;
                }

                case 7: {
                    // Logout
                    clearScreen();
                    cout << green << "Logging out.\n";
                    break;
                }

                default: {
                    cout << red << "Invalid choice. Please try again.\n" << white;
                    system("pause");
                }
            }

            // Exit the blog menu if the user chooses to log out
            if (blogChoice == 7) {
                break;
            }
        }
    }
}

    return 0;
}
```