|  |
| --- |
| **Department of Computer and Software Engineering–ITU** |
| **CE200L: Data Structures and Algorithms Lab** |

|  |  |
| --- | --- |
| **Course Instructor: Dr. Tahira Mahboob** | **Dated:** |
| **Lab Engineer: Muhammad Usama Riaz** | **Semester: 3rd** |
| **Session: 2022-2026** | **Batch: BSCE22** |

**Lab 13. Open Ended Lab**

|  |  |  |
| --- | --- | --- |
| **Name** | **Roll No** | **Total Marks**  **(Scaled out of 35)** |
|  |  |  |

Checked on:

Signature:

**Lab Task**

**Objective:**

In this lab, students are asked to select a specific data structure (e.g., linked list, binary tree, stack, queue) and applying it to solve a real-world problem of their choice. Through this exercise, students will gain a deeper understanding of data structures, their relevance in problem-solving, and the development of efficient algorithms.

**Tasks:**

1. **Problem Definition:**

Define the real-world problem that you intend to solve using the selected data structure. Provide context and background information to help readers understand the significance of the problem.

1. **Data Structure Selection and Relevance:**

Clearly specify the chosen data structure and its relevance to the selected problem. Discuss why the chosen data structure is suitable and how it aligns with the characteristics of the problem at hand.

1. **Algorithm and Method Development:**

Develop clear and concise algorithms for addressing the defined problem. Explain the steps involved and any decision-making processes within the algorithm.

Discuss the methods employed in your solution, considering factors like input validation, error handling, and overall robustness.

1. **Implementation:**

Implement the chosen data structure. Ensure that the implementation is well-structured, adheres to best coding practices, and includes necessary documentation**.**

Pay attention to aspects such as memory management, scalability, and code modularity.

1. **Analysis:**

Analyze the efficiency of your solution. Discuss time and space complexity, highlighting any trade-offs made during the design and implementation phases.

Evaluate the overall effectiveness of your solution in solving the specified problem. Consider edge cases and scenarios that might impact performance.

**Deliverables:**

1. **Code Files Submission (On GitHub)**. **Submission Deadline (11-12-2023)**
2. **Documentation/Report:(On Google Classroom & GitHub)**

Edit the current document to provide comprehensive documentation for your implementation, including code comments and summarizing your chosen data structure, its implementation, the defined problem, developed algorithms, and the analysis of your solution. **Submission Deadline (11-12-2023)**

1. **Presentation/Viva** (scheduled will be shared later).

**Group Members:**

**Syed Kifayat BSCE22026**

**Murtaza Khalid BSCE22004**

**Muhammad Haseeb BSCE22048**

**“Introduction of Lab”**

1.1 Purpose

The primary aim of this documentation is to serve as a comprehensive guide for the development and functionality of a C++-based social media console application. It is designed to provide in-depth insights into the core concepts of data structures and algorithms used in creating a social media platform reminiscent of Facebook. This documentation is intended to benefit both the developers actively involved in its creation and those seeking to understand and maintain the application.

1.2 Scope

This documentation covers an extensive scope, encompassing critical phases of the social media console application's development lifecycle. It begins from the conceptualization of the project, delves into the implementation of data structures and algorithms pertinent to a Facebook-like algorithm, and extends through the iterative stages of coding, testing, and refining the application. Each phase is meticulously detailed to provide a comprehensive understanding of how these core concepts are integrated into the application, fostering a deeper insight into its functionality and structure.

1.3 Audience

This documentation is crafted to cater to a diverse audience, including but not limited to:

**1.3.1 Development Team**:

This document serves as a valuable resource for the development team responsible for building and maintaining the website. It offers insights into the project's requirements, design, and technical details, aiding them in their tasks.

**1.3.2 Maintenance and Support Teams**:

Individuals entrusted with the continual maintenance and support of the social media console application can rely on this documentation for troubleshooting procedures, executing updates, and ensuring the seamless operation of the application.

**1.3.3 End Users**:

Although primarily tailored for development and maintenance stakeholders, this documentation also serves as a helpful reference for end users seeking insights into the functionalities and capabilities embedded within the social media console application. It offers a glimpse into the underlying mechanisms driving their user experience.

**“Requirement Gathering”**

2.1 Functional Requirements:

**2.1.1 Creating Account:**

Users can register for a new account by providing essential information such as username, email, and password. Upon successful registration, the system securely stores user credentials for subsequent logins.

**2.1.2 Show Feeds**:

Upon logging in, users are presented with a personalized feed displaying posts, updates, and activities from their friends or followed accounts. The feed is generated based on user interactions, preferences, and algorithmic recommendations.

**2.1.3 Add Friends**:

Users can send friend requests to other users, allowing them to connect and view each other's posts and updates. The recipient can either accept or decline the friend request.

**2.1.4 Seen Friend Request:**

Users are notified when their friend request has been seen by the recipient. This feature ensures transparency in the friend request process, indicating whether the recipient has viewed the request.

**2.1.5 Add Posts**:

Users can create and share posts consisting of text, images, videos, or links. They can customize the audience for each post, choosing to share it publicly, with friends, or specific groups.

**2.1.6 Friend Suggestion**:

The platform provides friend suggestions based on mutual friends, shared interests, or similar user interactions. These suggestions assist users in expanding their network within the platform.

**2.1.7 View Self Posts**:

Users have access to a profile section where they can view all their own posts, allowing them to review their activities and interactions within the platform.

2.2 Non-functional Requirements:

**2.2.1 Authentication:**

User authentication is a pivotal aspect of the application's security. It employs strong authentication mechanisms, including password hashing and encryption, to verify user identities during login.

**“Features and Functionality”**

3.1 Login System

The login system serves as the gateway for user access to the application and encompasses both the sign-up and sign-in options:

3.1.1 **Sign-Up Option:** Allows new users to create accounts by providing essential information such as username, email, password, and any additional required details. This feature facilitates the registration process, enabling new users to join the platform.

3.1.2  **Sign-In Option:** Enables existing users to securely log into their accounts using their credentials, typically username or email and password. This functionality grants access to the user's personalized content, social interactions, and other application features.

The login system ensures a secure and streamlined process for both new users seeking to join the platform and existing users aiming to access their accounts, fostering user engagement and interaction within the social media console application.

3.2 Home Page

Upon successful login, users are directed to the home page, tailored to their preferences and interactions within the application:

3.2.1 **Show Feeds:** The home page displays a dynamic feed, presenting a stream of posts, updates, and activities from users the individual is connected with. This feed algorithmically populates content based on user interactions, preferences, and relationships.

3.2.2 **Add Friends:** A dedicated section allows users to discover and connect with new friends. It enables users to expand their social network.

3.2.3 **Seen Friend Request:** Notifications or a designated section informs users about pending friend requests that they have seen but not yet accepted or declined, ensuring transparency in the friend request process.

3.2.4 **Add Post:** Users can create and share posts directly from the home page. This feature allows for the composition and publication of text or links, engaging with their social circle.

3.2.5 **View Self Posts:** A dedicated section or profile snapshot allows users to quickly view their own posts and interactions within the platform, providing easy access to their personal content and activities.

3.2.6 **Friend Suggestions:** Continuous suggestions of potential friends or connections based on mutual friends, common interests, or similar interactions within the platform, aiding users in expanding their social network.

**“Algorithm Overview”**

**4.1 User Management**

**4.1.1 User Class:** Define a class to store user details (name, ID, password, etc.).

**4.1.2 Graph Data Structure:** Use a graph to manage user relationships.

* + - Each node represents a user.
    - Edges denote friendships between users.
    - Adjacency lists can store friends' information.

**4.2 Authentication and Authorization**

**4.2.1 Implement user authentication:**

 Compare entered login details with stored user credentials.

**4.2.2 Authorization:** Define user roles and permissions if needed.

**4.3 Friend Requests**

* + - Store friend requests using queues or arrays.
    - Implement functions to send, receive, and manage friend requests.
    - Manage pending requests and allow users to accept or reject them.

**4.4 Posts and Feeds**

* + - Each user has a list of posts they've created.
    - Create a feed for each user showing their friends' recent posts.
    - Implement functions to add, delete, and view posts.
    - Update feeds when new posts are added.

**4.5 Suggestions**

 Implement a friend suggestion algorithm based on mutual connections.

**4.6 Implementation Using Data Structures**

## 4.6.1 Graph Representation

* Use an adjacency list or adjacency matrix to represent friendships.
* Store users in an array or hash map for quick access.

**4.6.2 Trie Data Structure for Auto-Complete (if needed)**

* Use a Trie to implement auto-complete functionalities like searching for users, names, etc.

**4.6.3 Queue/LinkedList for Friend Requests**

* Store pending friend requests using a queue or linked list.

## 4.6.4 Post Management

 Doubly linked lists or arrays to manage user posts.  Stacks or linked lists for managing feed updates.

**4.7 Algorithm Details**

## 4.7.1 User Registration and Login

* Read user data from files or databases during initialization.
* Authenticate users during login using stored credentials.

## 4.7.2 Friend Requests

* Allow users to send and accept/reject friend requests.
* Update the graph structure accordingly.

## 4.7.3 Post Creation and Feeds

* Enable users to create posts, manage their posts, and view their feeds.
* Update feeds based on friends' activities.

**4.7.4 Friend Suggestions**

* Implement a suggestion algorithm based on mutual friends or common interests.

**4.7.5 Auto-Complete for Searching**

* Utilize a Trie data structure for efficient auto-completion during user search.

**4.7.6 File Handling**

* Read and write data to files for user details, posts, friend requests, etc.

**4.8 Conclusion**

This algorithm provides a general framework for implementing a basic Facebook-like system using C++ and DSA concepts. Actual implementation details may vary based on specific requirements, optimizations, and additional features.

void afterLogin(GRAPHPTR graph, string userName)

{

    // Find the ID/index of the logged-in user

    int id = findIndex(graph, userName);

    int flag = 0;

    User \*currentUser = &graph->users[id];

    int choice;

    // Loop for user interaction options

    while (flag == 0)

    {

        system("clear");

        // Display menu options

        cout << "\n\t\t\t----------- Welcome to Facebook " << currentUser->name << " ----------- \n1).View your feed\n2).Post\n3).View your posts\n4).View all your friend requests\n5).View all your friends\n6).Send friend requests\n7).View suggested friends\n0).Exit\nEnter your choice : ";

        cin >> choice;

        switch (choice)

        {

        case 1:

        {

            system("clear");

            viewFeed(graph, id); // Display user's feed

            break;

        }

        case 2:

        {

            system("clear");

            addPost(graph, currentUser, "", 0); // Add a post

            break;

        }

        case 3:

        {

            system("clear");

            viewPosts(currentUser); // View user's posts

            break;

        }

        case 4:

        {

            system("clear");

            displayFriendRequests(graph, id); // View friend requests

            break;

        }

        case 5:

        {

            system("clear");

            viewFriends(currentUser); // View user's friends

            break;

        }

        case 6:

        {

            system("clear");

            sendFriendRequest(graph, currentUser->name, ""); // Send friend requests

            break;

        }

        case 7:

        {

            system("clear");

            friendSuggestions(graph, id); // View friend suggestions

            break;

        }

        case 0:

        {

            cout << "Thank you !\n";

            flag = 1; // Exit the loop and function

            break;

        }

        default:

        {

            cout << "Invalid choice\n"; // Handle invalid inputs

        }

        }

        cout << "\nPress Enter to continue....\n";

        getchar();

        getchar();

    }

}

// Function to manage the main login/signup page for Facebook

// Arguments:

//   - graph: Pointer to the graph structure containing user data

void main\_facebook\_login\_page(GRAPHPTR graph)

{

    int Check = 0;

    while (Check == 0)

    {

        system("clear");

        // Display the Facebook logo and login/signup options

        cout << "\n\t\t\t\t\tFacebook\n";

        cout << "\n\t\t----------Welcome to SignUp/SignIn Page----------\n\n1).SignUp\n2).SignIn\n0).Exit  \nYour Choice:";

        int Choice;

        cin >> Choice;

        switch (Choice)

        {

        case 1:

        {

            system("clear");

            cout << "\n\t\t----------Welcome to SignUp Page----------\n\n";

            string name, pass;

            cout << "Enter Name:";

            cin >> name;

            cout << "\nEnter Password:";

            cin >> pass;

            createAccount(graph, name, pass); // Create a new account

            break;

        }

        case 2:

        {

            system("clear");

            cout << "\n\t\t----------Welcome to SignIn Page----------\n";

            string userName, pass;

            cin.ignore();

            cout << "\nEnter Your User Name: ";

            getline(cin, userName);

            transform(userName.begin(), userName.end(), userName.begin(), ::tolower); // Convert username to lowercase

            cout << "\nEnter Your Password: ";

            getline(cin, pass);

            transform(pass.begin(), pass.end(), pass.begin(), ::tolower); // Convert password to lowercase

            bool check = loginAccount(graph, userName, pass); // Check login credentials

            if (check == true)

            {

                afterLogin(graph, userName); // Perform actions after successful login

            }

            else

            {

                cout << "\nInvalid login!\n"; // Handle invalid login

            }

            break;

        }

        case 0:

        {

            Check = 1;

            cout << "\n--------------Good Bye---------------\n\n";

            break;

        }

        default:

        {

            cout << "\nInvalid Choice!\n"; // Handle invalid inputs

        }

        }

        cout << "\nPress Enter to continue....\n";

        getchar();

        getchar();

    }

}

**And remaining code in the repository……….**

Assessment Rubric for Lab

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Performance metric** | **Task** | **CLO** | **Description** | **Max marks** | **Exceeds expectation** | **Meets expectation** | **Does not meet expectation** | **Obtained marks** |
|  |  |  |  |  | Executes without errors | Executes without errors, | Does not execute due to syntax errors, runtime errors, user prompts are misleading or non- existent. No testing has been completed (0-19) |  |
| 1. Realization |  |  |  |  | excellent user prompts, | user prompts are |
| of experiment | 1 | 1 | Functionality | 40 | good use of symbols, | understandable, minimum |
|  |  |  |  |  | spacing in output. Through | use of symbols or spacing in |
|  |  |  |  |  | testing has been completed (35-40) | output. Some  testing has been completed |
|  |  |  |  |  |  | (20-34) |
| 2. Teamwork | 1 | 3 | Group Performance | 5 | Actively engages and cooperates with other group member(s) in effective | Cooperates with other group member(s) in a reasonable manner but conduct can be | Distracts or discourages other group members from conducting the experiment (0-1) |  |
|  |  |  |  |  | manner (4-5) | improved (2-3) |  |
| 3. Conducting experiment | 1 | 1 | On Spot Changes | 10 | Able to make changes (8-10) | Partially able to make changes (5-7) | Unable to make changes (0-4) |  |
| 1 | 1 | Viva | 10 | Answered all questions (8-10) | Few incorrect answers (5-7) | Unable to answer all questions (0-4) |  |
| 4. Laboratory safety and disciplinary  rules | 1 | 3 | Code commenting | 5 | Comments are added and does help the reader to understand the code (4-5) | Comments are added and does not help the reader to understand the code (2-3) | Comments are not added (0-1) |  |
|  |  |  |  |  | Excellent use of white |  |  |  |
|  |  |  |  |  | space, creatively organized | Includes name, and | Poor use of white space |
| 5. Data  collection | 1 | 3 | Code Structure | 5 | work, excellent use of  variables and constants, | assignment, white space  makes the program fairly | (indentation, blank lines)  making code hard to read, |
|  |  |  |  |  | correct identifiers for constants, No line-wrap (4-5) | easy to read. Title, organized work, good use of variables  (2-3) | disorganized and messy (0-1) |
| 6. Data analysis | 1 | 4 | Algorithm | 20 | Solution is efficient, easy to  understand, and maintain (15-20) | A logical solution that is easy  to follow but it is not the most efficient (6-14) | A difficult and inefficient solution (0-5) |  |
| 7. Computer use | 1 | 2 | Documentation  & GitHub Submissions | 5 | Timely (4-5) | Late (2-3) | Not done (0-1) |  |
|  | Max Marks (total): | | | 100 | Obtained Marks (total): | | |  |