UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

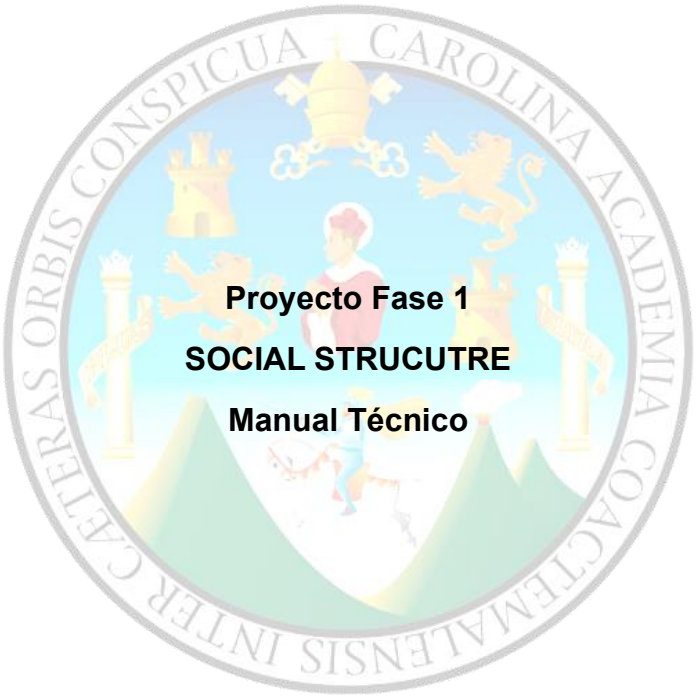
ESCUELA DE CIENCIAS Y SISTEMAS

Estructuras de datos B

Ing. Álvaro Hernández

Aux. Carlos Castro

Abdiel Fernando José Otzoy Otzín



Proyecto Fase 1

SOCIAL STRUCUTRE

Manual Técnico

Guatemala, Agosto 2024

Introducción

Este manual proporciona una guía detallada sobre cómo utilizar el sistema de red social implementado en C++. El sistema permite a los usuarios registrarse, iniciar sesión, gestionar amistades, publicar mensajes y ver reportes de actividad. Además, incluye funciones especiales para administradores, como la eliminación de usuarios y la generación de informes detallados. Este manual está diseñado para ayudar tanto a usuarios finales como a administradores a navegar y utilizar las funcionalidades del sistema de manera eficiente.

Requerimientos del Sistema

Para ejecutar el programa de la red social, es necesario cumplir con los siguientes requerimientos de hardware y software:

Requerimientos de Hardware

- **Procesador:** Mínimo de 1 GHz, recomendado 2 GHz o superior.
- **Memoria RAM:** Mínimo de 1 GB, recomendado 2 GB o más.
- **Espacio en disco:** Mínimo de 100 MB disponible para instalación y datos temporales.

Notas adicionales

- **Permisos de escritura:** Asegúrese de que el programa tenga permisos de escritura en el directorio actual para poder guardar y cargar archivos de datos (usuarios, solicitudes de amistad, publicaciones).
- **Archivos de configuración:** Si se utilizan archivos de configuración o datos iniciales, asegúrese de que estén en el formato correcto y ubicados en los directorios especificados.

Descripción general

auth.cpp es un archivo fuente de C++ que contiene las funciones de autenticación y gestión de usuarios para un sistema de red social. Implementa funcionalidades básicas como registro, inicio de sesión, cierre de sesión, envío de solicitudes de amistad, manejo de publicaciones y generación de reportes. Utiliza varias estructuras de datos para gestionar eficientemente los usuarios, sus relaciones y publicaciones.

Estructuras de datos utilizadas

1. **LinkedList** (Lista enlazada simple):
    - **Descripción:** Estructura de datos lineal donde cada elemento (nodo) contiene un puntero al siguiente nodo. Se utiliza para almacenar una colección de usuarios.

    - **Uso en auth.cpp:** Almacena todos los usuarios registrados. Permite la inserción, eliminación y búsqueda de usuarios.
2. **DoublyList** (Lista doblemente enlazada):
    - **Descripción:** Similar a la lista enlazada simple, pero cada nodo tiene punteros tanto al nodo siguiente como al nodo anterior. Esto permite una navegación bidireccional.
    - **Uso en auth.cpp:** Almacena todas las publicaciones. Permite agregar y listar publicaciones.
3. **Matrix** (Matriz de adyacencia):
    - **Descripción:** Estructura bidimensional que representa relaciones o conexiones entre elementos. Ideal para manejar relaciones en un grafo.
    - **Uso en auth.cpp:** Almacena las relaciones de amistad entre los usuarios. Cada relación de amistad es representada por una conexión entre dos nodos (usuarios).
4. **DoublyCircleList** (Lista circular doblemente enlazada):
    - **Descripción:** Variante de la lista doblemente enlazada donde el último nodo está conectado al primero, formando un ciclo cerrado.
    - **Uso en auth.cpp:** Almacena temporalmente las publicaciones disponibles para el usuario conectado y sus amigos. Facilita la navegación cíclica a través de las publicaciones.
5. **Stack** (Pila de solicitudes):
    - **Descripción:** estructura que almacena las solicitudes en un estructura tipo LIFO (Las In First Out)

Funciones

1. **addTestData()**
    - **Descripción:** Agrega datos de prueba al sistema, incluyendo usuarios y solicitudes de amistad.
    - **Retorno:** Ninguno.
    - **Mejoras:** Podría recibir parámetros para personalizar los datos de prueba o cargar desde un archivo de configuración.
2. **registerUser()**
    - **Descripción:** Registra un nuevo usuario en el sistema solicitando sus datos. Inserta el usuario en la lista enlazada y lo establece como usuario conectado.
    - **Retorno:** Ninguno.
    - **Mejoras:** Validar la entrada del usuario (por ejemplo, verificar si el correo ya está registrado).
3. **deleteUser()**
    - **Descripción:** Elimina un usuario del sistema. Si es administrador, puede eliminar a cualquier usuario por su correo; de lo contrario, el usuario elimina su propia cuenta.
    - **Retorno:** Ninguno.
    - **Mejoras:** Confirmación antes de eliminar, manejo de excepciones si el usuario no se encuentra.

4. **loginUser()**
    - **Descripción:** Permite a un usuario iniciar sesión proporcionando su correo y contraseña. Autentica al usuario y actualiza las variables globales isLogged e isAdmin.
    - **Retorno:** Ninguno.
    - **Mejoras:** Implementar un sistema de bloqueo tras múltiples intentos fallidos de inicio de sesión.
5. **logoutUser()**
    - **Descripción:** Cierra la sesión del usuario actual. Limpia las publicaciones disponibles y restablece las variables globales.
    - **Retorno:** Ninguno.
    - **Mejoras:** Confirmación antes de cerrar sesión si hay datos no guardados.
6. **logoutAdmin()**
    - **Descripción:** Cierra la sesión del administrador.
    - **Retorno:** Ninguno.
    - **Mejoras:** Similar a logoutUser(), con la opción de guardar el estado actual.
7. **sendRequest()**
    - **Descripción:** Permite al usuario conectado enviar una solicitud de amistad a otro usuario buscándolo por correo.
    - **Retorno:** Ninguno.
    - **Mejoras:** Verificar si ya existe una solicitud pendiente o si ya son amigos.
8. **respondRequest()**
    - **Descripción:** Permite al usuario responder a una solicitud de amistad. La respuesta puede ser aceptar o rechazar la solicitud.
    - **Retorno:** Ninguno.
    - **Mejoras:** Mostrar todas las solicitudes pendientes y permitir al usuario elegir cuál responder.
9. **createPost()**
    - **Descripción:** Crea una nueva publicación para el usuario conectado, solicitando el contenido de la publicación y generando una marca de tiempo.
    - **Retorno:** Ninguno.
    - **Mejoras:** Validar contenido de la publicación y manejar entradas grandes o vacías.
10. **viewProfile()**
    - **Descripción:** Muestra el perfil del usuario conectado.
    - **Retorno:** Ninguno.
    - **Mejoras:** Mostrar una vista más detallada del perfil, incluyendo publicaciones recientes o amigos en común.
11. **viewMyFriends()**
    - **Descripción:** Muestra la lista de amigos del usuario conectado.
    - **Retorno:** Ninguno.
    - **Mejoras:** Posibilidad de buscar en la lista de amigos o ver detalles específicos de un amigo.
12. **viewAvailablePosts()**

- **Descripción:** Muestra todas las publicaciones disponibles para el usuario conectado, incluidas las propias y las de sus amigos.
- **Retorno:** Ninguno.
- **Mejoras:** Implementar paginación o filtros para manejar grandes volúmenes de publicaciones.

13. **deleteAccount()**
    - **Descripción:** Elimina la cuenta del usuario conectado.
    - **Retorno:** Ninguno.
    - **Mejoras:** Confirmación antes de eliminar, y posibilidad de recuperación de cuenta.
14. **loadUsers()**
    - **Descripción:** Carga usuarios desde un archivo JSON y los inserta en la lista enlazada.
    - **Retorno:** Ninguno.
    - **Mejoras:** Verificar formato de datos y manejo de excepciones.
15. **loadRequests()**
    - **Descripción:** Carga solicitudes de amistad desde un archivo JSON y las procesa para los usuarios.
    - **Retorno:** Ninguno.
    - **Mejoras:** Manejo de errores y confirmación de carga exitosa.
16. **loadPosts()**
    - **Descripción:** Carga publicaciones desde un archivo JSON y las agrega a la lista doblemente enlazada de publicaciones.
    - **Retorno:** Ninguno.
    - **Mejoras:** Verificar contenido de publicaciones y manejo de errores.
17. **viewUsers()**
    - **Descripción:** Genera un archivo .dot con la lista de usuarios para visualización gráfica.
    - **Retorno:** Ninguno.
    - **Mejoras:** Confirmación de generación exitosa y ubicación del archivo.
18. **viewMatrix()**
    - **Descripción:** Genera un archivo .dot de la matriz de relaciones de amistad.
    - **Retorno:** Ninguno.
    - **Mejoras:** Verificar la coherencia de datos antes de la generación del archivo.
19. **viewPosts()**
    - **Descripción:** Genera un archivo .dot con la lista de publicaciones.
    - **Retorno:** Ninguno.
    - **Mejoras:** Similar a viewUsers(), confirmar generación y especificar ruta.
20. **viewTopFiveUsersWithMostPosts()**
    - **Descripción:** Muestra los cinco usuarios con más publicaciones.
    - **Retorno:** Ninguno.
    - **Mejoras:** Manejar empates y mostrar más detalles.
21. **viewTopFiveUsersWithLeastFriends()**
    - **Descripción:** Muestra los cinco usuarios con menos amigos.

    - **Retorno:** Ninguno.
    - **Mejoras:** Similar a la anterior, manejar empates y detalles adicionales.
22. **viewSentRequests()**
    - **Descripción:** Muestra las solicitudes de amistad enviadas por el usuario conectado.
    - **Retorno:** Ninguno.
    - **Mejoras:** Posibilidad de cancelar solicitudes enviadas.
23. **viewReceivedRequests()**
    - **Descripción:** Muestra las solicitudes de amistad recibidas por el usuario conectado.
    - **Retorno:** Ninguno.
    - **Mejoras:** Permitir aceptar/rechazar múltiples solicitudes al mismo tiempo.
24. **viewAvailablePostsDotFile()**
    - **Descripción:** Genera un archivo .dot de las publicaciones disponibles para el usuario conectado.
    - **Retorno:** Ninguno.
    - **Mejoras:** Confirmar generación y especificar ruta del archivo.

Mejoras generales sugeridas

- **Manejo de errores y excepciones:** Implementar un manejo de excepciones robusto para capturar y manejar errores inesperados.
- **Validación de entrada del usuario:** Validar todas las entradas de los usuarios para evitar inconsistencias y errores en los datos.
- **Interfaz de usuario:** Mejorar la interfaz del usuario para hacerla más intuitiva y amigable, con mensajes claros y navegación sencilla.
- **Documentación adicional:** Agregar comentarios en el código para explicar bloques de lógica complejos y su propósito.
- **Optimización de rendimiento:** Evaluar y optimizar las operaciones de búsqueda e inserción en las estructuras de datos para mejorar la eficiencia.

Documentación de main.cpp

Descripción general

main.cpp es el punto de entrada del programa y maneja la interacción con el usuario. Muestra un menú con opciones basadas en el estado del usuario (sin sesión, sesión de administrador, sesión de usuario normal). Llama a las funciones de auth.cpp según la opción seleccionada por el usuario.

Funciones principales

1. **main()**
    - **Descripción:** Función principal que inicializa el sistema y presenta un menú interactivo para el usuario. Dependiendo del estado de sesión

(usuario conectado o no, y si es administrador), presenta diferentes opciones y llama a las funciones correspondientes de auth.cpp.
    - **Retorno:** Entero (0 al finalizar exitosamente).
    - **Mejoras:** Implementar un bucle de control más robusto para manejar entradas inválidas y proporcionar un feedback más claro al usuario.
2. **showMenu()**
    - **Descripción:** Presenta las opciones del menú al usuario según su estado de sesión.
    - **Retorno:** Ninguno.
    - **Mejoras:** Refactorizar para reducir duplicación de código y mejorar la legibilidad.

Mejoras generales sugeridas

- **Modularidad:** Dividir el main() en funciones más pequeñas y modulares para mejorar la legibilidad y mantenibilidad.
- **Validación de entrada:** Verificar todas las entradas del usuario en los menús para evitar entradas no válidas.
- **Mensajes de ayuda:** Proveer mensajes de ayuda o instrucciones claras en cada opción del menú.
- **Feedback al usuario:** Confirmar al usuario cuando se realicen acciones exitosamente o cuando ocurra un error.