Universidad San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y sistemas
Estructuras de Datos

Ingenieros:

- Ing. Luis Espino
- Ing. Edgar Ornelis
- Ing. Álvaro Hernández

Auxiliares:

- Kevin Martinez
- Carlos Castro
- José Montenegro

MANUAL TECNICO

1. Introducción

Este manual técnico proporciona una guía detallada sobre la arquitectura, componentes, y funcionamiento interno del sistema de simulación de una red social desarrollado en C++. Está destinado a desarrolladores y técnicos que deseen entender, mantener, o ampliar el código fuente.

---

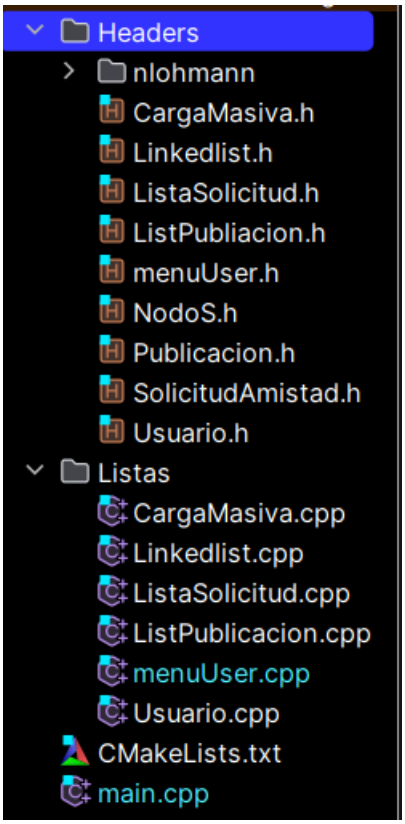
2. Requerimientos del Sistema

- Sistema Operativo: Windows 7 o superior
- Compilador: GCC/G++ versión 7.5 o superior
- Entorno de Desarrollo: Cualquier IDE o editor de texto que soporte C++ (Visual Studio Code, CLion, Dev-C++)
- Bibliotecas:
    - <iostream>
    - <windows.h>
    - <limits>
    - <string>
    - Json:nlohmann

3. Estructura del Proyecto

El proyecto está estructurado en los siguientes directorios y archivos clave:

- /src: Contiene los archivos fuente .cpp de la aplicación.
    - main.cpp: Punto de entrada principal del programa.
    - menuUser.cpp: Implementa el menú para usuarios registrados.
    - menuAdmin.cpp: Implementa el menú para administradores.
- /Headers: Contiene los archivos de cabecera .h para la definición de clases y funciones.
    - Linkedlist.h: Define la estructura y métodos para la lista enlazada de usuarios.
    - Usuario.h: Define la clase Usuario y sus métodos.
    - ListPubliacion.h: Define la lista doblemente enlazada para publicaciones.
    - menuUser.h: Declaraciones de funciones para el menú de usuario.
- /data: (Opcional) Directorio sugerido para almacenar archivos de datos como usuarios, relaciones y publicaciones.



4. Descripción de las Clases y Métodos

4.1 Clase Usuario

Ubicación: Headers/Usuario.h

Descripción: Esta clase representa a un usuario en la red social. Gestiona la información personal, solicitudes de amistad y publicaciones.

Atributos:

- int id: Identificador único del usuario.
- string nombre: Nombre del usuario.
- string apellidos: Apellidos del usuario.
- string fechaNacimiento: Fecha de nacimiento del usuario.
- string correo: Correo electrónico del usuario.
- string contraseña: Contraseña del usuario.
- vector<Usuario*> solicitudesRecibidas: Almacena las solicitudes de amistad recibidas.

```
12
13  struct Usuario {
14      int id;
15      string nombre;
16      string apellido;
17      string fechaN;
18      string correoE;
19      string pass;
20      Usuario* siguiente;
21      ListaSolicitud solicitudesRecibidas;
22      ListaSolicitud solicitudesEnviadas;
23
24      bool enviarSolicitud(Usuario*emisor,  Usuario* receptor);
25      void aceptarSolicitud();
26      void rechazarSolicitud();
27      void mostrarSolicitudMasReciente() const;
28
29      // Constructor para inicializar
30      Usuario(int id, string name, string apelli, string fechaNa, string mail, string pass);
31
32  };
33
34
35  #endif //USUARIOS_H
```

Métodos principales:

- void enviarSolicitud(Usuario* emisor, Usuario* receptor): Envía una solicitud de amistad.
- void aceptarSolicitud(): Acepta la solicitud de amistad más reciente.
- void rechazarSolicitud(): Rechaza la solicitud de amistad más reciente.
- void mostrarSolicitudMasReciente(): Muestra la solicitud de amistad más reciente.
- void perfil(string correo): Muestra la información del perfil del usuario.

```cpp
bool Usuario::enviarSolicitud(Usuario* emisor ,Usuario* receptor) {
    // Verificar si ya existe una solicitud pendiente o enviada
    if (solicitudesEnviadas.existeSolicitud(emisor, receptor) ||
        receptor->solicitudesRecibidas.existeSolicitud(emisor, receptor)) {
        std::cout << "Ya existe una solicitud pendiente o enviada a " << receptor->nombre << ".\n";
        return false;
        }

    // Crear nueva solicitud de amistad
    SolicitudAmistad* nuevaSolicitud = new SolicitudAmistad(emisor, receptor);
    solicitudesEnviadas.agregarSolicitud(nuevaSolicitud);
    receptor->solicitudesRecibidas.agregarSolicitud(nuevaSolicitud);
    std::cout << "Solicitud de amistad enviada a " << receptor->nombre << ".\n";
    return true;
}
```

```cpp

// Metodo para aceptar la solicitud de amistad más reciente
void Usuario::aceptarSolicitud() {
    if (!solicitudesRecibidas.cabeza) {
        std::cout << "No hay solicitudes de amistad para aceptar.\n";
        return;
    }

    // Aceptar la solicitud más reciente (al inicio de la lista)
    NodoSolicitud* solicitudAAceptar = solicitudesRecibidas.cabeza;
    solicitudesRecibidas.cabeza = solicitudesRecibidas.cabeza->siguiente;

    // Eliminar la solicitud de la lista del emisor
    solicitudAAceptar->solicitud->emisor->solicitudesEnviadas.eliminarSolicitud(solicitudAAceptar->solicitud);

    std::cout << "Solicitud de amistad aceptada.\n";

    delete solicitudAAceptar->solicitud; // Liberar memoria de la solicitud
    delete solicitudAAceptar; // Liberar memoria del nodo
}
```

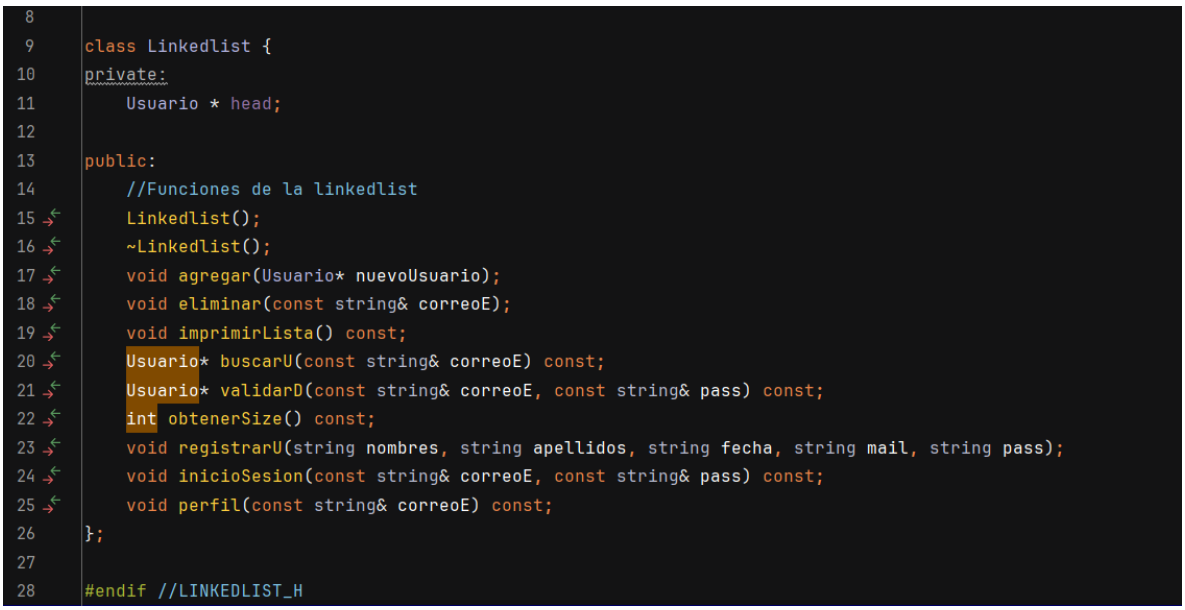
4.2 Clase Linkedlist

Ubicación: Headers/Linkedlist.h

Descripción: Implementa una lista enlazada simple para gestionar los usuarios registrados en el sistema.

Atributos:

- Usuario* head: Puntero al primer nodo de la lista.



```
8
9   class Linkedlist {
10  private:
11      Usuario * head;
12
13  public:
14      //Funciones de la linkedlist
15      Linkedlist();
16      ~Linkedlist();
17      void agregar(Usuario* nuevoUsuario);
18      void eliminar(const string& correoE);
19      void imprimirLista() const;
20      Usuario* buscarU(const string& correoE) const;
21      Usuario* validarD(const string& correoE, const string& pass) const;
22      int obtenerSize() const;
23      void registrarU(string nombres, string apellidos, string fecha, string mail, string pass);
24      void inicioSesion(const string& correoE, const string& pass) const;
25      void perfil(const string& correoE) const;
26  };
27
28  #endif //LINKEDLIST_H
```

Métodos principales:

- void agregar(Usuario* nuevoUsuario): Agrega un nuevo usuario a la lista.
- Usuario* buscarU(string correo): Busca un usuario en la lista por correo electrónico.
- void eliminar(string correo): Elimina un usuario de la lista.
- void inicioSesion(string correo, string contraseña): Valida las credenciales y permite el inicio de sesión.
- void imprimirLista(): Imprime todos los usuarios en la lista.

```cpp
void Linkedlist::agregar(Usuario* nuevoUsuario) { //Agrega nuevo usuario directamente sin validar nada
    nuevoUsuario->siguiente = head;
    head = nuevoUsuario;
}

void Linkedlist::eliminar(const string& correoE) {
    Usuario* actual = head;
    Usuario* anterior = nullptr;
    while (actual != nullptr && actual->correoE != correoE) {
        anterior = actual;
        actual = actual->siguiente;
    }
    if (actual != nullptr) {
        if (anterior == nullptr) {
            head = actual->siguiente;
        } else {
            anterior->siguiente = actual->siguiente;
        }
        delete actual;
    }
}
```

```cpp
Usuario* Linkedlist::buscarU(const string& correoE) const {
    Usuario* actual = head;
    while (actual != nullptr) {
        //Buscar usuario mediante su correo
        if (actual->correoE == correoE) {
            return actual;
        }
        actual = actual->siguiente;
    }
    return nullptr;
}

void Linkedlist::perfil(const string& correoE) const{
    Usuario* actual = head;
    while (actual != nullptr) {
        if (actual->correoE == correoE) { //Muestra el perfil del usuario actual
            cout << "Mis Datos: \n" " Nombre: " << actual->nombre << " " << actual->apellido << " " << "Cumpleanos: " << actual->fechaN
        }
        actual = actual->siguiente;
    }
}

int Linkedlist::obtenerSize() const {
    int size = 0;
    Usuario* actual = head;
    while (actual != nullptr) {
        size++;
        actual = actual->siguiente;
    }
    return size;
```

4.3 Clase ListaPublicaciones

Ubicación: Headers/ListPubliacion.h

Descripción: Implementa una lista doblemente enlazada para gestionar las publicaciones de los usuarios.

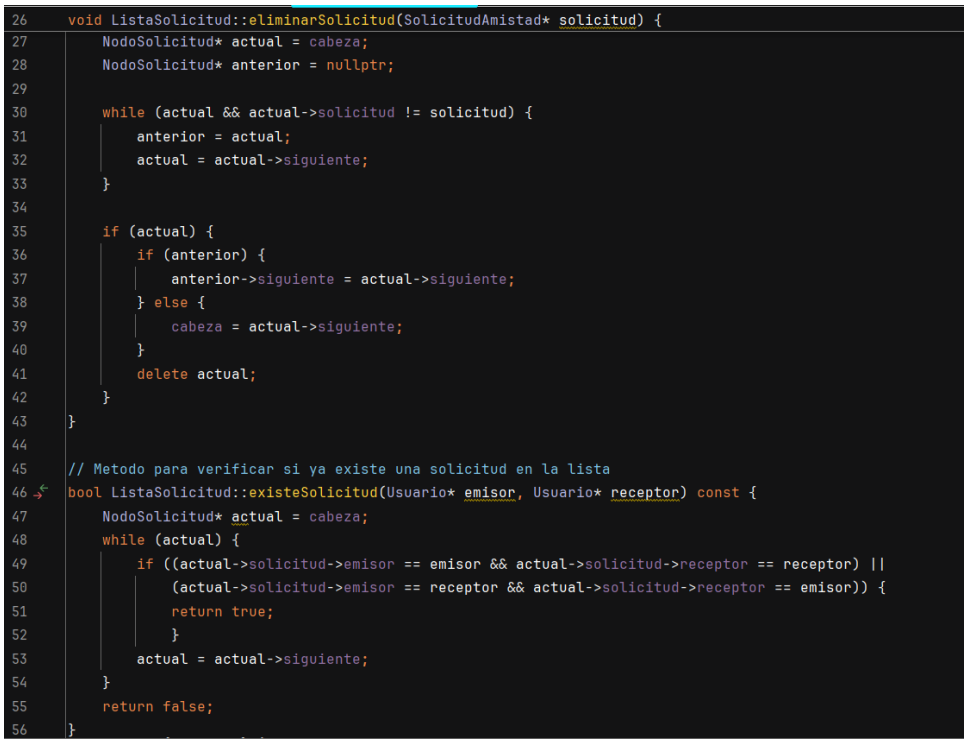
Atributos:

- Publicacion* head: Puntero al primer nodo de la lista de publicaciones.

```
9
10  class ListaSolicitud {
11
12  public:
13      NodoSolicitud* cabeza;
14      ListaSolicitud();
15      void agregarSolicitud( SolicitudAmistad* solicitud);
16      void eliminarSolicitud(SolicitudAmistad* solicitud);
17      bool existeSolicitud(Usuario* emisor, Usuario* receptor) const;
18  };
19
20  #endif //LISTASOLICITUD_H
```

Métodos principales:

- void agregarPublicacion(string correo, string contenido): Agrega una nueva publicación a la lista.
- void eliminarPublicacion(string correo): Elimina una publicación de la lista.
- void navegarPublicaciones(): Permite navegar entre las publicaciones en orden cronológico.



```
void ListaSolicitud::eliminarSolicitud(SolicitudAmistad* solicitud) {
    NodoSolicitud* actual = cabeza;
    NodoSolicitud* anterior = nullptr;

    while (actual && actual->solicitud != solicitud) {
        anterior = actual;
        actual = actual->siguiente;
    }

    if (actual) {
        if (anterior) {
            anterior->siguiente = actual->siguiente;
        } else {
            cabeza = actual->siguiente;
        }
        delete actual;
    }
}

// Metodo para verificar si ya existe una solicitud en la lista
bool ListaSolicitud::existeSolicitud(Usuario* emisor, Usuario* receptor) const {
    NodoSolicitud* actual = cabeza;
    while (actual) {
        if ((actual->solicitud->emisor == emisor && actual->solicitud->receptor == receptor) ||
            (actual->solicitud->emisor == receptor && actual->solicitud->receptor == emisor)) {
            return true;
            }
        actual = actual->siguiente;
    }
    return false;
}
```

---

5. Lógica del Programa

5.1 Flujo del Menú Principal

1. Inicio del Programa: El programa inicia y muestra el menú principal mediante la función mostrarMenu.
2. Selección de Opciones: Según la selección del usuario (iniciar sesión, registrarse, mostrar información, salir), se ejecutan diferentes bloques de código en el switch principal.
3. Iniciar Sesión: Se valida el correo y la contraseña ingresados. Si son correctos, se redirige al menuUser correspondiente.
4. Registro: Los datos ingresados por el usuario se capturan y se crean nuevos objetos Usuario, que se añaden a la Linkedlist de usuarios.

5.2 Manejo de Usuarios

- Registro de Usuarios: Los usuarios son añadidos a la Linkedlist mediante el método agregar. Cada nodo de la lista enlazada contiene un objeto Usuario.
- Eliminación de Usuarios: Al eliminar un usuario, se recorre la lista enlazada, se busca el correo proporcionado, y se elimina el nodo correspondiente.

5.3 Manejo de Publicaciones

- Agregar Publicación: Se crea un nuevo nodo en la lista doblemente enlazada ListaPublicaciones, vinculando la publicación con el correo del usuario.
- Navegación de Publicaciones: El usuario puede navegar hacia adelante y hacia atrás en la lista doblemente enlazada, mostrando el contenido de las publicaciones.

---

6. Funcionalidades Avanzadas

6.1 Módulo de Administración

- Carga Masiva: Permite la carga de usuarios, relaciones y publicaciones desde archivos externos mediante las funciones cargaUsuarios, cargaSolicitud, y cargaPubli.
- Gestión de Usuarios: Permite al administrador eliminar usuarios del sistema a través del menú de administrador.