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 describe la arquitectura, la lógica de implementación y los componentes del software de una aplicación desarrollada en C++ utilizando la biblioteca de Qt. El proyecto consiste en la simulación de una red social con gestión de usuarios, publicaciones y comentarios, y utiliza estructuras de datos complejas como árboles AVL, listas doblemente enlazadas, y un árbol binario de búsqueda (ABB).

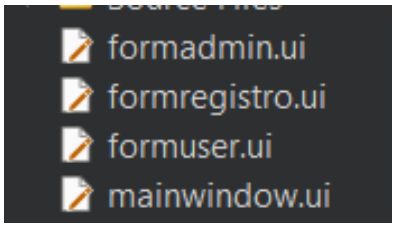
2. Requisitos Técnicos

- **Lenguaje**: C++ (C++17 o superior recomendado)
- **Framework**: Qt 6.x (para la interfaz gráfica)
- **Sistema Operativo**: Windows o cualquier otro sistema que soporte Qt y MinGW o equivalente.
- **Compilador**: MinGW64-bit
- **Entorno de Desarrollo**: Qt Creator

3. Arquitectura del Sistema

La aplicación tiene una arquitectura basada en ventanas que gestionan las interfaces de usuarios y administradores:

1. **Ventana Principal (MainWindow)**:
    - Proporciona opciones de inicio de sesión y registro.
    - Identifica al usuario y carga la interfaz correspondiente (usuario o administrador).
2. **Ventana de Usuario (formUser)**:
    - Utilizada para gestionar las acciones del usuario, como crear publicaciones, comentarios y acceder a la información personal.
3. **Ventana de Administrador (formAdmin)**:
    - Gestiona el acceso a la administración global de la aplicación, como usuarios y estadísticas.
4. **Ventana de Registro (formregistro)**:
    - Permite a los nuevos usuarios registrarse y ser añadidos al árbol AVL de usuarios.



4. Estructuras de Datos Utilizadas

4.1 Árbol AVL (arbolAVL)

- **Propósito**: Almacenar y gestionar información sobre los usuarios.
- **Implementación**: El árbol AVL mantiene los usuarios ordenados por correo electrónico, lo cual facilita búsquedas, inserciones y eliminaciones balanceadas.
- **Métodos Importantes**:
    - insertU(user*): Inserta un nuevo usuario en el árbol.
    - eliminar(const std::string&): Elimina un usuario en base al correo electrónico.

- searchU(const std::string&): Busca un usuario y retorna un puntero al objeto user.

```cpp
#ifndef ARBOLAVL_H
#define ARBOLAVL_H
#include "nodoavl.h"

class arbolAVL
{
private:

    NodoAVL* raiz;
    int height(NodoAVL* node);
    NodoAVL* insert(NodoAVL* node, user* u);
    int getBalance(NodoAVL* node);
    NodoAVL* rotateRight(NodoAVL* y);
    NodoAVL* rotateLeft(NodoAVL* x);
    NodoAVL* minValueNode(NodoAVL* node);
    NodoAVL* deleteNode(NodoAVL* root, const std::string& correoE);
    std::string preOrder(NodoAVL* node);
    std::string inOrder(NodoAVL* node);
    std::string postOrder(NodoAVL* node);
    void generarDot(NodoAVL* node, std::ofstream& out);


public:
    arbolAVL() : raiz(nullptr) {}
    void insertU(user* usuario);
    bool search(const std::string& correoE);
    NodoAVL* searchNode(NodoAVL* node, const std::string& correoE);
    bool credenciales(const std::string& correoE, const std::string& contrasena);
    std::string mostrarInfo(const std::string& correoE);
    user* searchU(const std::string& correoE);
    void eliminar(const std::string& correoE);
    std::string mostrarPreOrder();
    std::string mostrarInOrder();
    std::string mostrarPostOrder();
    void graficar(const std::string& nombreArchivo);
};

#endif // ARBOLAVL_H

```

4.2 Lista Doble Enlazada (ListaDoble)

- **Propósito**: Gestionar las publicaciones de los usuarios y sus comentarios.
- **Implementación**: Cada nodo contiene una publicación y la lista permite navegar entre publicaciones hacia adelante y hacia atrás.
- **Métodos Importantes**:
    - insertarFinal(...): Inserta una publicación al final de la lista.
    - eliminarPublicacionMasReciente(const std::string&): Elimina la publicación más reciente de un usuario específico.
    - buscarPorFecha(const std::string&): Busca publicaciones por fecha.

```cpp
#ifndef LISTADOBLE_H
#define LISTADOBLE_H

#include "nodolistadoble.h"

class ListaDoble
{
private:
    nodoListadoble* cabeza;
    nodoListadoble* cola;

public:
    ListaDoble();
    ~ListaDoble();
    void insertarFinal(std::string correo, std::string contenido, std::string fecha, std::string hora, std::string Imagen);
    std::string mostrarAdelante();
    std::string mostrarAtras();
    std::string buscarPorFecha(std::string fecha);
    void eliminarPublicacion(std::string correo, std::string fecha);
    void agregarComentario(std::string correo, std::string fecha, std::string hora, Comentario comentario);
    std::string mostrarComentarios(std::string fecha, std::string hora);
    std::string mostrarPublicaciones();
    std::string buscarCorreoPorHora(std::string fecha, std::string hora);
    void generarGraphviz(std::string nombreArchivo);
    int contarPublicaciones();
    std::vector<std::pair<nodoListadoble*, int>> obtenerPublicacionesConComentarios();
    std::string top3PublicacionesConMasComentarios();

    bool eliminarPublicacionMasReciente(std::string correo);

    // Eliminar todas las publicaciones de un usuario
    void eliminarPublicacionesPorCorreo(std::string correo);


    nodoListadoble* obtenerCabeza() {
        return cabeza;  // Retornar el puntero al primer nodo
    }
};

#endif // LISTADOBLE_H

```

4.3 Árbol Binario de Búsqueda (ABB)

- **Propósito**: Almacenar publicaciones de acuerdo con la fecha de creación, permitiendo consultas eficientes.
- **Implementación**: Las fechas son utilizadas como clave para organizar las publicaciones en un ABB.
- **Métodos Importantes**:
    - insertarPublicacion(...): Inserta una nueva publicación en el árbol.
    - mostrarInordenLimitado(int), mostrarPreordenLimitado(int), mostrarPostordenLimitado(int): Permiten mostrar un número limitado de publicaciones según el tipo de recorrido.
    - top3FechasConMasPublicaciones(): Retorna las tres fechas con más
    - publicaciones.

```cpp
#ifndef ABB_H
#define ABB_H

#include "NodoABB.h"
#include "vector"


class ABB
{

private:

    NodoABB* raiz;
    NodoABB* insertarRecursivo(NodoABB* nodo, std::string fecha);
    NodoABB* buscarNodo(NodoABB* nodo, std::string fecha);
    void mostrarPublicacionesPorFecha(NodoABB* nodo, std::string fecha);
    std::string inordenLimitado(NodoABB* nodo, int& contador, int limite);
    std::string preordenLimitado(NodoABB* nodo, int& contador, int limite);
    std::string postordenLimitado(NodoABB* nodo, int& contador, int limite);
    std::vector<std::pair<std::string, int>> contarPublicacionesPorFecha(NodoABB* nodo);
    static bool compararPorCantidad(const std::pair<std::string, int>& a, const std::pair<std::string, int>& b);
    NodoABB* buscarNodoPorFecha(NodoABB* nodo, const std::string& fecha);
    void generarGraphvizRec(NodoABB* nodo, std::ofstream& archivo, std::string fechaEspecifica);
    void eliminarPublicacionesPorCorreoRec(NodoABB* nodo,std::string correo);


public:

    ABB() {
        raiz = nullptr;
    }

    bool eliminarPublicacion(std::string correo);
    void eliminarPublicacionesPorCorreo(std::string correo);

    void insertarPublicacion(std::string correo, std::string contenido, std::string fecha, std::string hora, std::string pathImagen);
    void mostrarPublicacionesPorFecha(std::string fecha);
    std::string mostrarInordenLimitado(int limite);
    std::string mostrarPreordenLimitado(int limite);
    std::string mostrarPostordenLimitado(int limite);
    std::string top3FechasConMasPublicaciones();
    void graficarABBConListaFechaEspecifica(std::string nombreArchivo, std::string fechaEspecifica);

};

#endif // ABB_H

```

4.4 Árbol B de Orden 5 (Comentarios)

- **Propósito:** Almacenar los comentarios realizados en publicaciones, ordenándolos por correo, fecha y hora.
- **Implementación:** El árbol B de orden 5 permite almacenar y buscar los comentarios de manera eficiente.

```cpp
#ifndef ARBOLB_H
#define ARBOLB_H
#include "nodocomentario.h"
#include "Comentario.h"



class ArbolB
{

private:
    nodoComentario* raiz;
    int orden;

    void insertarNoLleno(nodoComentario* nodo, Comentario comentario);
    void dividirHijo(nodoComentario* nodoPadre, int indice);
    int compararFechaHora(const Comentario& c1, const Comentario& c2);
    std::string imprimirInorden(nodoComentario* nodo);
    int contarComentariosRecursivo(nodoComentario* nodo);

public:
    ArbolB(int _orden) {
        raiz = nullptr;
        orden = _orden;
    }

    void insertar(Comentario comentario);
    std::string imprimir();
    int contarComentarios();

};

#endif // ARBOLB_H

```

5. Descripción del Código

5.1 Archivos de Cabecera y Dependencias

- mainwindow.h y ui_mainwindow.h: Define la ventana principal, responsable de gestionar la interfaz gráfica del inicio de sesión.
- formuser.h, formadmin.h, formregistro.h: Define las interfaces y funcionalidades para cada tipo de usuario.
- QMessageBox y QFileDialog: Utilizados para mostrar mensajes al usuario y abrir archivos (imágenes para publicaciones).

5.2 Ususarios :

```cpp
#ifndef USER_H
#define USER_H

#include <string>

class user{
    private:
    std::string nombre;
    std::string apellido;
    std::string fechaN;
    std::string correoE;
    std::string pass;

public:

// Constructor del usuario
user(const std::string& nombre, const std::string& apellido, const std::string& fechaN, const std::string& correoE, const std::string& pass);

    std::string getnombre() const;
    std::string getapellido() const;
    std::string getfechaN() const;
    std::string getcorreoE() const;
    std::string getpass() const;

    void setnombre(const std::string& nombre);
    void setapellido(const std::string& apellido);
    void setfechaN(const std::string& fechaN);
    void setcorreoE(const std::string& correoE);
    void setpass(const std::string& pass);
    std::string  mostrarInfo();
    std::string  mostrarInfoG();
};

#endif // USER_H
```

6 Grafo:

Sustituyendo la matriz dispersa ahora se cuenta con una nueva estructura (Grafo no dirigido) para la gestión de relaciones de amistad y sugerencias.

```cpp
#ifndef GRAPH_H
#define GRAPH_H

#include "nodoG.h"
#include <string>
#include <vector>
class Graph
{

private:

    Node* head;  // Cabeza de la lista de usuarios
    std::vector<std::string> users;  // Lista de correos electrónicos para referencia por índice
    Node* findNode(int value);


public:
    Graph() : head(nullptr) {}

    void addUser(const std::string& email);
    int getUserIndex(const std::string& email);
    void addFriendship(const std::string& email1, const std::string& email2);
    std::string displayFriends(const std::string& email);
    std::string suggestFriends(const std::string& email);
    bool isDirectFriend(Node* user, int target);

    void generateAdjacencyListGraph(const std::string& filename);
    void generateCompleteGraph(const std::string& filename);
    void generateHighlightedGraph(const std::string& filename, const std::string& email);
};

#endif // GRAPH_H

```

```cpp
#ifndef NODOG_H
#define NODOG_H

#include <fstream>

// Clase para los vecinos (amistades) de un nodo
class Neighbor {
public:
    int value;  // Índice del usuario en el grafo
    Neighbor *next;

    Neighbor(int val) : value(val), next(nullptr) {}
};

// Clase para los nodos (usuarios) del grafo
class Node {
public:
    int value;  // Índice del usuario
    Node* next;  // Siguiente usuario
    Neighbor* neighbors;  // Lista de amigos (vecinos)

    Node(int val) : value(val), next(nullptr), neighbors(nullptr) {}

    // Insertar un vecino (amistad)
    void insertNeighbor(int destiny) {
        Neighbor* newNeighbor = new Neighbor(destiny);
        newNeighbor->next = neighbors;
        neighbors = newNeighbor;
    }

    // Mostrar los vecinos (amistades)
    void graphNeighbors(std::ofstream &out) {
        Neighbor* current = neighbors;
        while (current != nullptr) {
            out << current->value << " ";
            current = current->next;
        }
    }
};

```