Manual Técnico

Social Structure

Fecha: 21 de septiembre de 2024

Contenido

OBJETIVO

El manual técnico proporciona una guía detallada para desarrolladores, administradores de sistemas y otros usuarios técnicos sobre cómo instalar, configurar y utilizar Social Structure. Contiene información sobre la estructura del código, las dependencias del proyecto, las instrucciones de implementación, y la explicación de los diferentes componentes y su funcionamiento.

ALCANCE

El proyecto "Social Structure" es un sistema de gestión de redes sociales que permite la administración y visualización de usuarios, solicitudes de amistad, y publicaciones. Está desarrollado en C++ y hace uso de varias estructuras de datos, como listas enlazadas simples, listas enlazadas dobles, pilas, árboles AVL, BST y B para organizar y gestionar la información de manera eficiente.

DESARROLLO DEL MANUAL TÉCNICO

Características Principales

- Gestión de Usuarios: Los usuarios pueden registrarse, iniciar sesión, y ver su perfil. Los administradores tienen la capacidad de cargar usuarios desde archivos JSON, así como eliminar usuarios existentes.
- Solicitudes de Amistad: Los usuarios pueden enviar y recibir solicitudes de amistad. Las solicitudes recibidas se almacenan en pilas asociadas a cada usuario receptor y se pueden gestionar desde su perfil. Asimismo, las solicitudes enviadas se almacenan en listas simples del usuario emisor.
- Publicaciones: Los usuarios pueden crear, visualizar, y filtrar publicaciones. Las publicaciones se gestionan utilizando listas enlazadas dobles y árboles binarios de búsqueda BST, lo que permite una organización eficiente y la posibilidad de reportes.
- Reportes: El sistema permite generar reportes en dentro de la interfaz gráfica para visualizar la estructura de datos de los usuarios, sus relaciones, y publicaciones. Además, los administradores pueden ver un Top 5 de los usuarios con más publicaciones.
- Menú Administrativo: Los administradores tienen acceso a un menú exclusivo donde pueden gestionar usuarios, relaciones, y publicaciones a través de cargas masivas desde archivos JSON.
- Autenticación: El sistema incluye un mecanismo de autenticación para garantizar que solo los usuarios registrados y el administrador tengan acceso a sus respectivas funciones.

Tecnologías utilizadas

- C++: El proyecto está desarrollado principalmente en lenguaje C++, aprovechando las bibliotecas nativas para implementar la funcionalidad requerida.
- Graphviz: Se utiliza la biblioteca Graphviz para generar gráficos visuales que representan las mascotas y su estado.
- Qt: IDE utilizado para ejecutar la interfaz gráfica.

Requisitos del sistema

- Compilador C++
- Sistema operativo compatible con C++

- Graphviz
- Qt

Instalación

Debido a que es una aplicación que se ejecuta en consola, se debe clonar o descargar el repositorio del proyecto desde https://github.com/angelygm03/-EDD-Proyecto_202210483.git para luego abrir la carpeta Fase 2 del proyecto en el IDE de su preferencia (se recomienda Qt) y correr la aplicación.

Estructura del proyecto

El proyecto sigue una estructura de directorios típica de C++

Fase 1

Fase 2

- Main.cpp: Script principal para ejecutar la aplicación.
- Mainwindow: Ventana del login
- Adminwindow: Ventana de administrador
- Userwindow: Ventana de usuario
- AVLTree: Estructura para almacenar a los usuarios
- BinarySearchTree: Estructura que almacena las publicaciones de un usuario
- DoubleList: Estructura que almacena las publicaciones globalmente
- Stack: Estructura que almacena las solicitudes recibidas
- Solicitud List: Estructura que almacena las solicitudes enviadas
- Friend List: Estructura que almacena las relaciones de amistad
- Fecha List: Estructura que almacena las publicaciones por fecha

Estructuras utilizadas

Listas Enlazadas

El proyecto ha implementado el uso de tres listas enlazadas simples, una para almacenar las publicaciones por fecha, otra para las relaciones de amistad y la última para las solicitudes de amistad enviadas.

```
class SolicitudNode {
public:
    string emisor;
    string receptor;
    string estado;
    SolicitudNode* next;

    SolicitudNode(string e, string r, string s) : emisor(e), receptor(r), estado(s), next(nullptr) {}
};

class SolicitudList {
private:

public:
    SolicitudNode* head;
    int size;
    SolicitudList();
    void insert(string emisor, string receptor, string estado);
    void print();
    bool eliminar(const string& emisor, const string& receptor);
    SolicitudNode* buscar(const string& emisor, const string& receptor);
    void generateDotFile(const string& fileName);
    bool eliminarPorEmisorYReceptor(const string& emisor, const string& receptor);
};

```

Lista doble

Como se mencionó anteriormente, esta estructura almacena las publicaciones de formar global en todo el sistema, solamente el usuario administrador puede acceder a ella.

```
class DoubleList {
private:
    NodeDoubleList* head;
    NodeDoubleList* tail;
    int size;

public:
    DoubleList();

    void insertAtBeginning(string correo, string contenido, string fecha, string hora, string imagenPath);
    void insertAtEnd(const string& correo, const string& contenido, const string& fecha, const string& hora, const string& imagenPath);
    void insertAtPosition(int givenPosition, string correo, string contenido, string fecha, string hora, string imagenPath);
    void print() const;
    bool deleteAtPosition(int position);
    void printByUser(const string& correoUsuario) const;
    bool deleteByUser(int position, const string& correoUsuario);
    void generateDotFile();
    void printTopUsersByPublications() const;
};
```

Pila

La pila fue una buena elección para poder mostrarle las solicitudes pendientes al usuario, de esta manera cuando el usuario quiera ver y aprobar las solicitudes se le muestran primero las útlimas solicitudes entrantes.

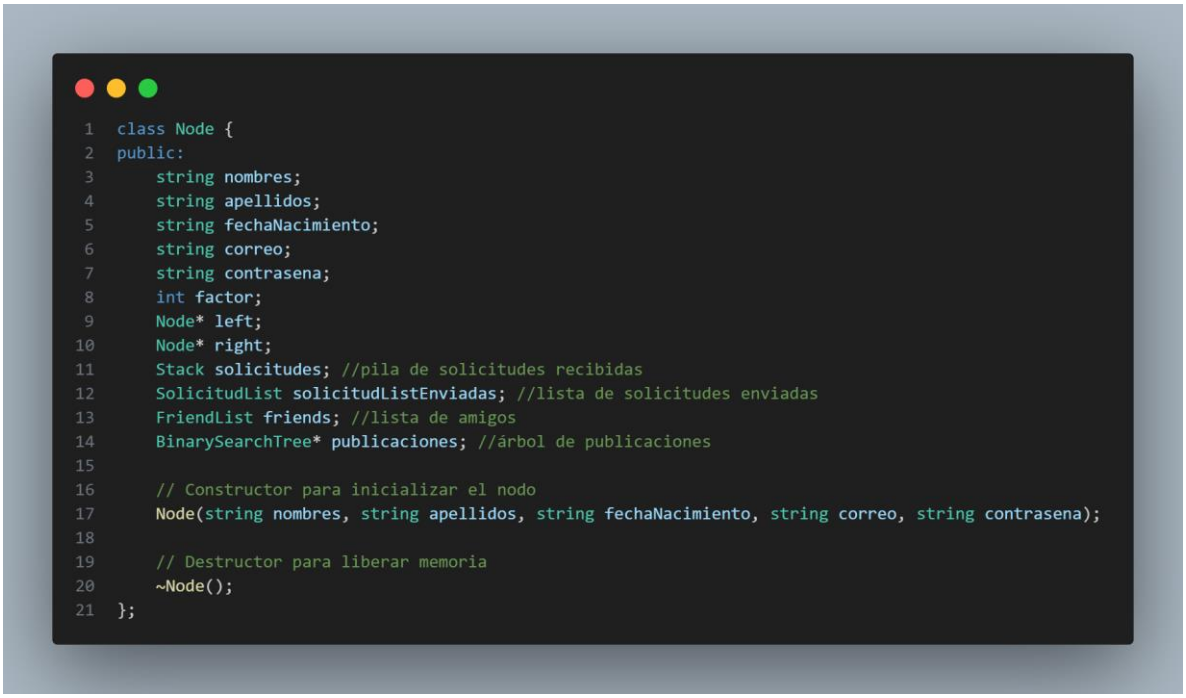
```
class StackNode {
public:
    string emisor;
    string receptor;
    string estado;
    StackNode* next;

    // Constructor del nodo
    StackNode(string emisor, string receptor, string estado);

};
```

Árbol AVL

Este árbol almacena los usuarios del sistema teniendo como criterio el correo electrónico de la A-Z.



```
class Node {
public:
    string nombres;
    string apellidos;
    string fechaNacimiento;
    string correo;
    string contrasena;
    int factor;
    Node* left;
    Node* right;
    Stack solicitudes; //pila de solicitudes recibidas
    SolicitudList solicitudListEnviadas; //lista de solicitudes enviadas
    FriendList friends; //lista de amigos
    BinarySearchTree* publicaciones; //árbol de publicaciones

    // Constructor para inicializar el nodo
    Node(string nombres, string apellidos, string fechaNacimiento, string correo, string contrasena);

    // Destructor para liberar memoria
    ~Node();
};
```

Árbol BST

Esta estructura permite la eficiencia en la búsqueda de publicaciones, por lo que cada nodo tiene como criterio las fechas de las publicaciones y cada nodo tiene su propia lista enlazada con las publicaciones para esa fecha.

```cpp
class NodeBST {
public:
    string fecha;  // Clave para el nodo del BST
    FechaList publicaciones;  // Lista de publicaciones para esa fecha

    NodeBST* left;
    NodeBST* right;

    NodeBST(string fecha);  // Constructor
};
```

Árbol B

El árbol B se ha implementado para almacenar los comentarios de las publicaciones. Es un árbol de orden 5.

```cpp
class NodoB {
public:
    Comentario** comentarios;   // Puntero a comentarios (hasta 4 comentarios por nodo)
    NodoB** hijos;              // Puntero a hijos (hasta 5 hijos por nodo)
    int numComentarios;         // Número actual de comentarios en el nodo
    bool hoja;                  // True si es un nodo hoja

    NodoB(bool esHoja);
    void insertarComentario(Comentario* comentario);
    void dividirNodo(int i, NodoB* y);
    void imprimir();
    void graficar(std::ofstream &file);
};

class ArbolB {
private:
    NodoB* raiz;

public:
    ArbolB();
    void insertar(Comentario* comentario);
    void imprimir();
    void graficar(const std::string &filename);
};
```