Introducción

El software "Social Structure" es una aplicación que permite a los usuarios registrarse, iniciar sesión y acceder a diversas funcionalidades sociales. Este manual técnico proporciona una guía detallada sobre la arquitectura, configuración, implementación y mantenimiento del software.

Arquitectura del Sistema

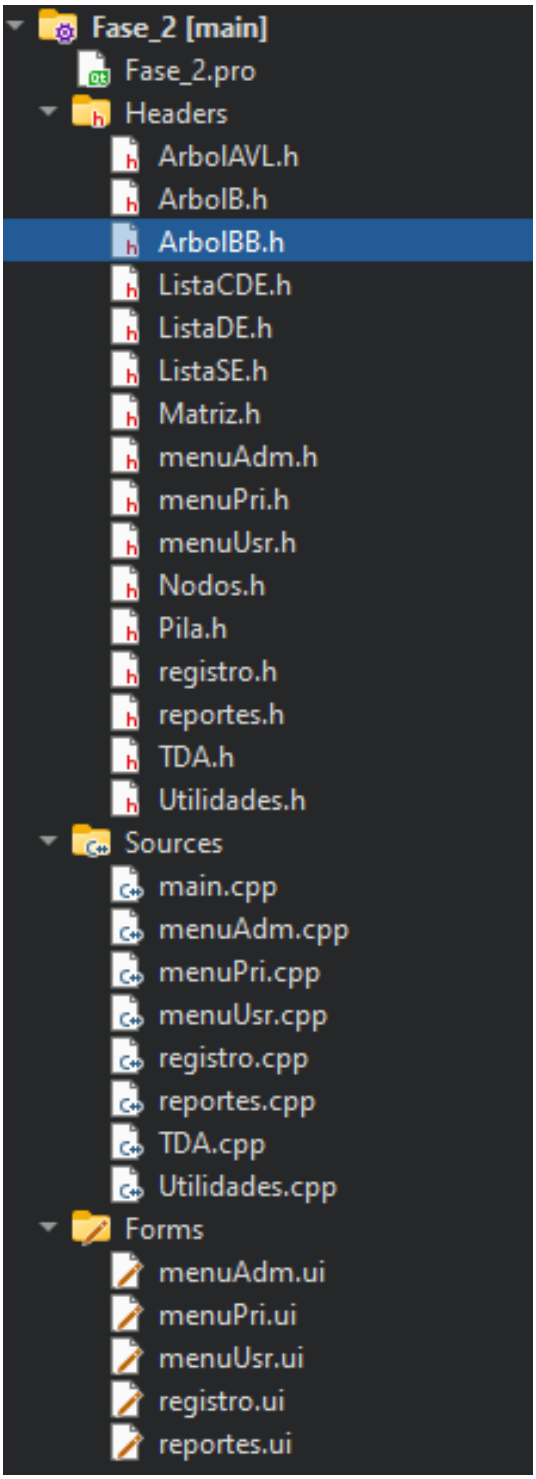
- **Componentes Principales**:
    - **Interfaz de Usuario**: Maneja la interacción con el usuario.
    - **Lógica del programa**: Contiene la lógica principal del software.
    - **Base de Datos**: Almacena la información de los usuarios.

Configuración del Entorno de Desarrollo

- **Requisitos del Sistema**:
    - Sistema operativo: Windows 10 o superior
    - IDE: Qt Creator.
    - Compilador: G++
- **Instrucciones de Configuración**:
    1. Descargar e instalar Qt: https://www.qt.io/download-qt-installer-oss.
    2. Instalar el kit Development para escritorio ya incluye el compilador mingw.
    3. Instalar las librerías de nlohmann/json: JSON for Modern C++ (github.com)
    4. Instalar la última versión de Graphviz
    5. Asegurarse que las variables de entorno del sistema (PATH) estén bien configuradas.
    6. Clonar el repositorio del proyecto:
       git clone https://github.com/TheSuppaOnion/EDD_Social-Structure_201708880
    7. Abrir el proyecto en Qt Creator.

Estructura del Código

- **Fotos estructura proyecto**:



Detalles de Implementación

- **Depuración**:
    - El código tiene comentada la mayoría de su funcionalidad para futuras revisiones.
    - Existen mensajes de error que se imprimen en la consola para más información en caso de errores.
- **Descripción de estructuras:** Todas las estructuras fueron hechas mediante templates para poderles asignar tipos, de esta forma se pueden crear estructuras de datos mas generales y no es necesario el especificar tanto en los header de estructuras. A continuación, se describe el archivo TDA.h el cual es la central de nuestra base de datos.
    - **TDA.h:** En este archivo se declaran las variables globales que no queremos perder información como los usuarios, publicaciones, etc. También definimos las estructuras pertinentes como Usuario, Publicación, Comentario, Relación y Solicitud. Estas nos ayudan a guardar datos en las estructuras de datos, todas las estructuras de datos están incluidas en la parte de arriba.

```cpp
#ifndef TDA_H
#define TDA_H

#include "ArbolAVL.h"
#include "ArbolBB.h"
#include "ArbolB.h"
#include "ListaSE.h"
#include "ListaDE.h"
#include "ListaCDE.h"
#include "Matriz.h"
#include "Pila.h"
#include <memory>
#include <vector>
#include <string>
#include <iostream>
#include <unordered_map>

using namespace std;

// ---------------------------------------------------- INICIO STRUCT ----------------------------------------------------

struct Usuario {
    string nombre;
    string apellido;
    string fechaNac;
    string correoElectronico;
    string contrasenia;
    Usuario(string nombre, string apellido, string fechaNac, string correoElectronico, string contrasenia) : nombre(nombre), apellido(apellido), fechaNac(fechaNac), correoElectronico(correoElectronico), contrasenia(contrasenia){}
    friend ostream& operator<<(ostream& os, const Usuario& u) {
        os << u.nombre << " " << u.apellido << " " << u.fechaNac << " "  << u.correoElectronico << " " << u.contrasenia << " ";
        return os;
    }
    string toString() const {
        return nombre + "," + apellido + "," + fechaNac + "," + correoElectronico + "," + contrasenia;
    }

    vector<string> ObtenerPartes() const {
        return {nombre, apellido, fechaNac, correoElectronico, contrasenia};
    }

    bool operator<(const Usuario& otro) const {
        return correoElectronico < otro.correoElectronico;
    }

    bool operator>(const Usuario& otro) const {
        return correoElectronico > otro.correoElectronico;
    }

    bool operator==(const Usuario& otro) const {
        return correoElectronico == otro.correoElectronico;
    }
    // Sobrecargar el operador > para comparar con std::string
    bool operator>(const string& other) const {
        return correoElectronico > other;
    }

    // Sobrecargar el operador < para comparar con std::string
    bool operator<(const string& other) const {
        return correoElectronico < other;
    }

    //Sobrecargar el operador == para comparar con std::string
```

```cpp
struct Comentario {
    string correo;
    string contenido;
    string fechayhora;
    // Constructor por defecto
    Comentario() : correo(""), contenido(""), fechayhora("") {}

    Comentario(string correo, string contenido, string fechayhora): correo(correo), contenido(contenido), fechayhora(fechayhora){}
    friend ostream& operator<<(ostream& os, const Comentario& c) {
        os << c.correo << " " << c.contenido << " " << c.fechayhora << " ";
        return os;
    }

    bool operator<(const Comentario& otro) const {
        return fechayhora < otro.fechayhora;
    }

    bool operator>(const Comentario& otro) const {
        return fechayhora > otro.fechayhora;
    }

    bool operator==(const Comentario& otro) const {
        return fechayhora == otro.fechayhora;
    }

    // Sobrecargar el operador > para comparar con std::string
    bool operator>(const string& other) const {
        return fechayhora > other;
    }

    // Sobrecargar el operador < para comparar con std::string
    bool operator<(const string& other) const {
        return fechayhora < other;
    }

    bool operator==(const string& other) const {
        return fechayhora == other;
    }

    string toString() const {
        return "Correo: " + correo + ", Contenido: " + contenido + ", Fecha y Hora: " + fechayhora; // Customize this to return the desired string representation
    }

    string ObtenerCorreo() const {
        return correo;
    }

    string ObtenerContenido() const {
        return contenido;
    }

    string ObtenerFechayHora() const {
        return fechayhora;
    }
};

struct Publicacion {
    string correo;
    string contenido;
    string fecha;
    string hora;
```

```cpp
struct Publicacion {
    string correo;
    string contenido;
    string fecha;
    string hora;
    ArbolB<Comentario> comentarios; // Árbol B que almacena comentarios indexados por fecha y hora

    Publicacion(string correo, string contenido, string fecha, string hora): correo(correo), contenido(contenido), fecha(fecha), hora(hora) {}
    friend ostream& operator<<(ostream& os, const Publicacion& p) {
        os << p.correo << " " << p.contenido << " " << p.fecha << " " << p.hora << " ";
        return os;
    }

    bool operator<(const Publicacion& otro) const {
        return fecha < otro.fecha;
    }

    bool operator>(const Publicacion& otro) const {
        return fecha > otro.fecha;
    }

    bool operator==(const Publicacion& otro) const {
        return fecha == otro.fecha;
    }

    // Sobrecargar el operador > para comparar con std::string
    bool operator>(const string& other) const {
        return fecha > other;
    }

    // Sobrecargar el operador < para comparar con std::string
    bool operator<(const string& other) const {
        return fecha < other;
    }

    bool operator==(const string& other) const {
        return fecha == other;
    }
    string ObtenerCorreo() const {
        return correo;
    }

    string ObtenerContenido() const {
        return contenido;
    }

    string ObtenerFecha() const {
        return fecha;
    }

    string ObtenerHora() const {
        return hora;
    }
    ArbolB<Comentario> ObtenerComentarios() const{
        return comentarios;
    }
};

struct Solicitud {
    string usuarioemisor;
    string correoElectronico;
    string fecha;
```

```
struct Solicitud {
    string usuarioemisor;
    string correoElectronico;
    string fecha;
    string hora;
    Solicitud(string usuarioemisor, string correoElectronico, string fecha, string hora) : usuarioemisor(usuarioemisor), correoElectronico(correoElectronico), fecha(fecha), hora(hora){}
    friend ostream& operator<<(ostream& os, const Solicitud& s) {
        os << s.usuarioemisor << " " << s.correoElectronico << " " << s.fecha << " " << s.hora << " ";
        return os;
    }
    vector<string> ObtenerPartes() const {
        return {usuarioemisor, correoElectronico, fecha, hora};
    }
    string ObtenerCorreo() const {
        return correoElectronico;
    }
};

struct Relaciones{
    string usuarioemisor;
    string usuarioreceptor;
    string estado;
    Relaciones(string usuarioemisor, string usuarioreceptor, string estado) : usuarioemisor(usuarioemisor), usuarioreceptor(usuarioreceptor), estado(estado){}
    friend ostream& operator<<(ostream& os, const Relaciones& r) {
        os << r.usuarioemisor << " " << r.usuarioreceptor << " " << r.estado << " ";
        return os;
    }
    vector<string> ObtenerPartes() const {
        return {usuarioemisor, usuarioreceptor, estado};
    }
};

// ------------------------------------------------- FINAL STRUCT -------------------------------------------------

// ------------------------------------------------- ESTRUCTURAS GLOBALES -------------------------------------------------

extern ArbolAVL<Usuario> listadmin;
extern ArbolAVL<Usuario> listausers;
extern ListaDE<Publicacion> listapub;
extern Matriz<Relaciones> matrizrel;
extern unordered_map<string, shared_ptr<ArbolBB<Publicacion>>> PublicacionesPorUsuario;
extern unordered_map<string, shared_ptr<ListaSE<Solicitud>>> SolicitudesEnviadasPorUsuario;
extern unordered_map<string, shared_ptr<Pila<Solicitud>>> SoolicitudesRecibidasPorUsuario;

// ------------------------------------------------- FINAL ESTRUCTURAS GLOBALES -------------------------------------------------

#endif // TDA_H
```

- **Nodos.h:** En este header declaramos todos los nodos necesarios para crear los valores necesarios en las estructuras de datos.

```
#ifndef NODOS_H
#define NODOS_H

#include <memory>

template <typename T>
class Nodo {
public:
    T dato;
    Nodo* siguiente;
    Nodo* anterior;
    Nodo(T dato) : dato(dato), siguiente(nullptr), anterior(nullptr) {}
};

template <typename T>
struct NodoPila {
    T dato;
    NodoPila<T>* siguiente;
    NodoPila(T dato) : dato(dato), siguiente(nullptr) {}
};

// Clase NodeVal para manejar los valores en la matriz
template <typename T>
class NodeVal {
public:
    bool exists = false;
    T dato;
};

// Clase Node para cada nodo de la matriz
template <typename T>
class NodoMatriz {
public:
    int i = -1, j = -1;
    T dato;
    NodoMatriz<T>* up = nullptr;
    NodoMatriz<T>* down = nullptr;
    NodoMatriz<T>* right = nullptr;
    NodoMatriz<T>* left = nullptr;

    NodoMatriz(int i = -1, int j = -1, T dato = T()) : i(i), j(j), dato(dato) {}
};

template <typename T>
class NodoAVL {
public:
    T dato;
    NodoAVL* izquierdo;
    NodoAVL* derecho;
    int altura;

    NodoAVL(T val) : dato(val), izquierdo(nullptr), derecho(nullptr), altura(1) {}
};

// Declaración adelantada de ListaSE
template <typename T>
class ListaSE;

template <typename T>
class NodoBB {
public:
    T dato;
```

```cpp
// Clase Nodo para cada nodo de la matriz
template <typename T>
class NodoMatriz {
public:
    int i = -1, j = -1;
    T dato;
    NodoMatriz<T>* up = nullptr;
    NodoMatriz<T>* down = nullptr;
    NodoMatriz<T>* right = nullptr;
    NodoMatriz<T>* left = nullptr;

    NodoMatriz(int i = -1, int j = -1, T dato = T()) : i(i), j(j), dato(dato) {}
};

template <typename T>
class NodoAVL {
public:
    T dato;
    NodoAVL* izquierdo;
    NodoAVL* derecho;
    int altura;

    NodoAVL(T val) : dato(val), izquierdo(nullptr), derecho(nullptr), altura(1) {}
};

// Declaración adelantada de ListaSE
template <typename T>
class listaSE;

template <typename T>
class NodoBB {
public:
    T dato;
    NodoBB* izquierdo;
    NodoBB* derecho;
    std::shared_ptr<ListaSE<T>> listaPublicaciones;

    NodoBB(T dato) : dato(dato), izquierdo(nullptr), derecho(nullptr), listaPublicaciones(std::make_shared<ListaSE<T>>()) {}
};

#define MAX 5
#define MIN 1

template <typename T>
class NodoB {
public:
    T val[MAX + 1];
    int num = 0;
    NodoB<T>* link[MAX + 1];
    bool hoja;

    NodoB(bool hoja) : hoja(hoja) {
        for (int i = 0; i < MAX + 1; i++) {
            link[i] = nullptr;
        }
    }
};

#endif // NODOS_H
```

- **ArbolAVL.h:** Todas las estructuras están definidas de la siguiente forma sino muy similares para mantener un orden y legibilidad. Este header en especial es para guardar los usuarios ya que para eso sirve el AVL.

```cpp
#ifndef ARBOLAVL_H
#define ARBOLAVL_H

#include "Nodos.h"
#include <fstream>
#include <vector>
#include <string>
#include <iostream>

using namespace std;

template <typename T>
class ArbolAVL {
private:
    NodoAVL<T>* raiz;
    int altura(NodoAVL<T>* nodo);
    int balanceFactor(NodoAVL<T>* nodo);
    void actualizarAltura(NodoAVL<T>* nodo);
    NodoAVL<T>* rotarDerecha(NodoAVL<T>* y);
    NodoAVL<T>* rotarIzquierda(NodoAVL<T>* x);
    NodoAVL<T>* balancear(NodoAVL<T>* nodo);
    NodoAVL<T>* Insertar(NodoAVL<T>* nodo, T dato);
    NodoAVL<T>* borrar(NodoAVL<T>* nodo, const string &dato);
    NodoAVL<T>* Encontrar(NodoAVL<T>* nodo, const string &dato);
    NodoAVL<T>* Buscar(NodoAVL<T>* nodo, const string& correo);
    void destruir(NodoAVL<T>* nodo);
    void ObtenerUsuarios(NodoAVL<T>* nodo, vector<string>& usuarios);
    void printRec(NodoAVL<T>* nodo, const string& nodeName, ofstream& file, int& nodeCount) const;
    void preorden(NodoAVL<T>* nodo, vector<T>& usuarios);
    void postorden(NodoAVL<T>* nodo, vector<T>& usuarios);
    void inorden(NodoAVL<T>* nodo, vector<T>& usuarios);
public:
    ArbolAVL(): raiz(nullptr){};
    ~ArbolAVL() {destruir(raiz);};
    void Insertar(T dato);
    bool Modificar(const string& correo, const T& nuevosDatos);
    vector<string> ObtenerUsuarios();
    bool validarLogin(const string& nombre, const string& contrasenia);
    bool arbolVacio();
    void borrar(const string& dt);
    bool Encontrar(const string& dt);
    NodoAVL<T>* Buscar(const string& correo);
    string renderGraphviz(const string& filename) const;
    void preorden(vector<T>& usuarios);
    void postorden(vector<T>& usuarios);
    void inorden(vector<T>& usuarios);
};
```

Mantenimiento:

- El código esta comentado en varios puntos para saber exactamente que hace cada función. Si ocurren errores o si se quiere implementar algo lo más probable seria revisar estos archivos