MANUAL DE USUARIO

KEVIN ANDRES ALVAREZ HERRERA

202203038

ESTRUCTURA DE DATOS

SECCION C

Social Structure

INDICE

Objetivos

Específicos

- Crear una red social para facilitar la interacción entre los usuarios a pesar de la distancia.
- Permitir la interacción de los usuarios y publicaciones realizadas por ellos y sus amigos.
- Desarrollar una plataforma de red social que permita a los usuarios conectarse y comunicarse con amigos y nuevos contactos, independientemente de su ubicación geográfica.

Generales

- Implementar funcionalidades que permitan a los usuarios realizar publicaciones, compartir contenido y participar en interacciones con otros usuarios de su red de contactos.
- Crear un sistema de gestión de usuarios que facilite el registro, autenticación y administración de perfiles, así como la búsqueda y conexión con otros usuarios.
- Diseñar mecanismos que promuevan la privacidad y seguridad de los usuarios, asegurando que la información compartida en la red social sea protegida y accesible solo para los contactos autorizados.

Especificación técnica

Requisitos de hardware

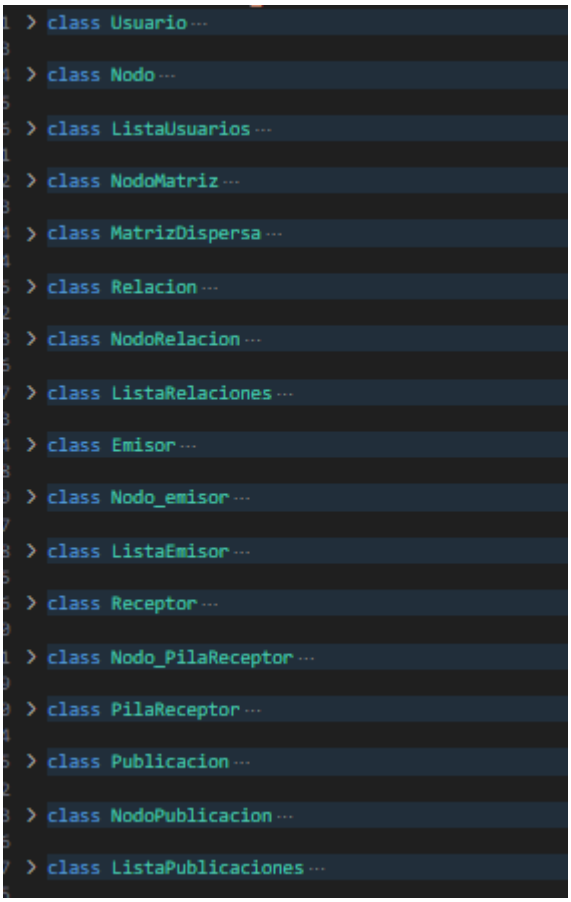
- Monitor
- Mouse
- Teclado
- CPU
- RAM
- Memoria de almacenamiento

Requisitos de software

- Sistema operativo Windows 10 o 11
- Herramientas como Visual Studio Code
- La última versión de c++ (13.2.0)

Lógica para la realización del programa

Clases utilizadas



Estructuras utilizadas

Listas enlazadas

Listas doblemente enlazadas

Listas circulares doblemente enlazadas

Pilas

Matriz dispersa

Explicación de codigo

```
class Usuario
{
public:
    Usuario(std::string nombre, std::string apellido, std::string fecha_de_nacimiento, std::string correo, std::string contrasena)···

    std::string getNombre() const { return nombre_; }
    std::string getApellido() const { return apellido_; }
    std::string getFechaDeNacimiento() const { return fecha_de_nacimiento_; }
    std::string getCorreo() const { return correo_; }
    std::string getContrasena() const { return contrasena_; }
    friend class ListaUsuarios;

public:
    std::string nombre_;
    std::string apellido_;
    std::string fecha_de_nacimiento_;
    std::string correo_;
    std::string contrasena_;

    friend class ListaUsuarios;
};

class Nodo
{
public:
    Usuario usuario;
    Nodo *siguiente;

    Nodo(const Usuario &usuario) : usuario(usuario), siguiente(nullptr)···
};
```

Métodos y Funcionalidades:

- *Constructor*: Inicializa los atributos del usuario con los valores proporcionados.
- *Getters*:
- *getNombre*(): Devuelve el nombre del usuario.
- *getApellido*(): Devuelve el apellido del usuario.
- *getFechaDeNacimiento*(): Devuelve la fecha de nacimiento del usuario.
- *getCorreo*(): Devuelve el correo electrónico del usuario.
- *getContrasena*(): Devuelve la contraseña del usuario.

```cpp
class Nodo
{
public:
    Usuario usuario;
    Nodo *siguiente;

    Nodo(const Usuario &usuario) : usuario(usuario), siguiente(nullptr)
    {
        std::cout << "Depuración: Nodo creado para usuario con correo: " << usuario.getCorreo() << std::endl;
    }
};

class ListaUsuarios
{
public:
    ListaUsuarios() : cabeza(nullptr) {}

    ~ListaUsuarios()
    {
        Nodo *actual = cabeza;
        while (actual != nullptr)
        {
            Nodo *temp = actual;
            actual = actual->siguiente;
            delete temp;
            temp = nullptr;
        }
    }

    void generateDot(const std::string &filename) const
    {
        std::ofstream file(filename);
        if (file.is_open())
        {
            file << "digraph G {" << std::endl;
            file << "node [shape=record];" << std::endl;
            file << "rankdir=LR;" << std::endl;

            Nodo *current = cabeza;
            int id = 0;
            while (current != nullptr)
            {
                file << "node" << id << " [label=\"{" << "Nombre: " << current->usuario.getNombre() << "\\n"
                     << "Correo: " << current->usuario.getCorreo() << "}\"];" << std::endl;
                if (current->siguiente != nullptr)
                {
                    file << "node" << id << " -> node" << (id + 1) << ";" << std::endl;
                }
                current = current->siguiente;
                id++;
            }

            file << "}" << std::endl;
            file.close();
            delete current;
            current = nullptr;
        }
        else
        {
            std::cerr << "No se pudo abrir el archivo" << std::endl;
        }
    }
```

```cpp
void renderGraphviz(const std::string &dotFilename, const std::string &imageFilename) const
{
    std::string command = "dot -Tpng " + dotFilename + " -o " + imageFilename;
    int result = system(command.c_str());
    if (result != 0)
    {
        std::cerr << "Error al generar la imagen con Graphviz" << std::endl;
    }
}

void agregarUsuario(const Usuario &usuario)
{
    std::cout << "Agregando usuario: " << usuario.getNombre() << std::endl;

    if (usuarioDuplicado(usuario.getCorreo()))
    {
        std::cerr << "Usuario con correo " << usuario.getCorreo() << " ya existe." << std::endl;
        return;
    }

    Nodo *nuevoNodo = nullptr;

    try
    {
        nuevoNodo = new Nodo(usuario);
    }
    catch (const std::bad_alloc &e)
    {
        std::cerr << "Error de asignación de memoria al agregar usuario: " << e.what() << std::endl;
        return;
    }

    if (cabeza == nullptr)
    {
        // Si la lista está vacía, el nuevo nodo es tanto la cabeza como la cola
        cabeza = nuevoNodo;
        cola = nuevoNodo;
        std::cout << "Usuario agregado como cabeza de la lista." << std::endl;
    }
    else
    {
        // Agregar el nuevo nodo después de la cola actual
        cola->siguiente = nuevoNodo;
        cola = nuevoNodo; // Actualizar la cola para que apunte al nuevo nodo
        std::cout << "Usuario agregado al final de la lista." << std::endl;
    }
}

bool usuarioDuplicado(const std::string &correo) const
{
    Nodo *actual = cabeza;
    while (actual != nullptr)
    {
        if (actual->usuario.getCorreo() == correo)
        {
            return true;
        }
        actual = actual->siguiente;
    }
    actual = nullptr;
    delete actual;
    return false;
};
```

```cpp
bool buscarUsuarioPorCorreoyContrasena(const std::string &correo, const std::string &contrasena) const
{
    Nodo *temp = cabeza;
    while (temp != nullptr)
    {
        if (temp->usuario.getCorreo() == correo && temp->usuario.getContrasena() == contrasena)
        {
            temp = nullptr;
            delete temp;
            return true;
        }
        temp = temp->siguiente;
    }
    return false;
};

bool buscarCorreo(const std::string &correo) const
{
    Nodo *temp = cabeza;
    while (temp != nullptr)
    {
        if (temp->usuario.getCorreo() == correo)
        {
            temp = nullptr;
            delete temp;
            return true;
        }
        temp = temp->siguiente;
    }
    return false;
};
```

```cpp
void borrarUsuarioPorCorreo(const std::string &correo)
{
    if (cabeza == nullptr)
    {
        std::cerr << "La lista está vacía." << std::endl;
        return;
    }

    if (cabeza->usuario.getCorreo() == correo)
    {
        std::cout << "Eliminando nodo con correo: " << correo << std::endl;
        Nodo *temp = cabeza;
        cabeza = cabeza->siguiente;

        if (cabeza == nullptr)
        {
            std::cout << "Lista está ahora vacía después de eliminar el nodo." << std::endl;
        }
        else
        {
            std::cout << "Nueva cabeza después de eliminar nodo: " << cabeza->usuario.getCorreo() << std::endl;
        }

        temp = nullptr;
        delete temp;
        return;
    }

    Nodo *actual = cabeza;
    Nodo *anterior = nullptr;

    while (actual != nullptr && actual->usuario.getCorreo() != correo)
    {
        std::cout << "Chequeando nodo con correo: " << actual->usuario.getCorreo() << std::endl;
        anterior = actual;
        actual = actual->siguiente;
    }

    if (actual == nullptr)
    {
        std::cerr << "No hay usuario con el correo " << correo << " que se pueda borrar." << std::endl;
        return;
    }

    if (anterior != nullptr)
    {
        std::cout << "Eliminando nodo con correo: " << actual->usuario.getCorreo() << std::endl;
        anterior->siguiente = actual->siguiente;
        return;
    }

    actual->siguiente = nullptr;

    delete actual;
    actual = nullptr;
    std::cout << "Usuario con correo " << correo << " ha sido borrado." << std::endl;

    return;
}
```

```cpp
void mostrarDatosPorCorreo(const std::string &correo) const
{
    Nodo *cabeza = this->cabeza;
    Nodo *temp = cabeza;
    bool encontrado = false;

    while (temp != nullptr)
    {
        if (temp->usuario.getCorreo() == correo)
        {
            std::cout << "Usuario encontrado:" << std::endl;
            std::cout << "Nombre: " << temp->usuario.getNombre() << std::endl;
            std::cout << "Apellido: " << temp->usuario.getApellido() << std::endl;
            std::cout << "Fecha de nacimiento: " << temp->usuario.getFechaDeNacimiento() << std::endl;
            std::cout << "Correo: " << temp->usuario.getCorreo() << std::endl;
            std::cout << "Contraseña: " << temp->usuario.getContrasena() << std::endl;
            encontrado = true;
            break;
        }
        temp = temp->siguiente;
    }

    if (!encontrado)
    {
        std::cerr << "No se encontró un usuario con el correo " << correo << "." << std::endl;
    }
}

std::string limpiarCadena(const std::string &str)
{
    std::string resultado = str;
    resultado.erase(0, resultado.find_first_not_of(" \n\r\t\""));
    resultado.erase(resultado.find_last_not_of(" \n\r\t\"") + 1);
    return resultado;
}

void cargarUsuariosDesdeJson(const std::string &nombreArchivo)
{
    std::ifstream archivo(nombreArchivo);

    if (!archivo.is_open())
    {
        std::cerr << "Error al abrir el archivo JSON." << std::endl;
        return;
    }

    try
    {
        nlohmann::json jsonData;
        archivo >> jsonData;

        for (const auto &item : jsonData)
        {
            std::string nombre = limpiarCadena(item.at("nombres").get<std::string>());
            std::string apellido = limpiarCadena(item.at("apellidos").get<std::string>());
            std::string fecha_de_nacimiento = limpiarCadena(item.at("fecha_de_nacimiento").get<std::string>());
            std::string correo = limpiarCadena(item.at("correo").get<std::string>());
            std::string contrasena = limpiarCadena(item.at("contraseña").get<std::string>());

            Usuario usuario(nombre, apellido, fecha_de_nacimiento, correo, contrasena);
            agregarUsuario(usuario);
        }
    }
    catch (const nlohmann::json::exception &e)
    {
        std::cerr << "Error al procesar el archivo JSON: " << e.what() << std::endl;
    }
```

```cpp
void registrarUsuario(ListaUsuarios &listaUsuarios)
{
    std::string nombre, apellido, fecha_de_nacimiento, correo, contrasena;

    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');

    std::cout << "Ingrese su nombre: ";
    std::getline(std::cin, nombre);

    if (!nombre.empty())
    {
        std::cout << "Ingrese su apellido: ";
        std::getline(std::cin, apellido);

        if (!apellido.empty())
        {
            std::cout << "Ingrese su fecha de nacimiento (DD/MM/AAAA): ";
            std::getline(std::cin, fecha_de_nacimiento);

            if (!fecha_de_nacimiento.empty())
            {
                std::cout << "Ingrese su correo: ";
                std::getline(std::cin, correo);

                if (!correo.empty())
                {
                    if (!listaUsuarios.usuarioDuplicado(correo))
                    {
                        std::cout << "Ingrese su contraseña: ";
                        std::getline(std::cin, contrasena);
                        // Crear el nuevo usuario
                        Usuario nuevoUsuario(nombre, apellido, fecha_de_nacimiento, correo, contrasena);

                        // Agregarlo a la lista de usuarios
                        listaUsuarios.agregarUsuario(nuevoUsuario);

                        std::cout << "Usuario registrado exitosamente." << std::endl;
                    }
                    else
                    {
                        std::cerr << "El correo ya está registrado." << std::endl;
                    }
                }
                else
                {
                    std::cerr << "El correo no puede estar vacío." << std::endl;
                }
            }
            else
            {
                std::cerr << "La fecha de nacimiento no puede estar vacía." << std::endl;
            }
        }
        else
        {
            std::cerr << "El apellido no puede estar vacío." << std::endl;
        }
    }
    else
    {
        std::cerr << "El nombre no puede estar vacío." << std::endl;
    }
}
```

```
private:
    Nodo *cabeza;
    Nodo *cola;
};
```

Clase Nodo

Atributos:

Usuario usuario: Almacena un objeto de la clase Usuario.

*Nodo *siguiente*: Un puntero que apunta al siguiente nodo en la lista enlazada.

Constructor:

Nodo(const Usuario &usuario): Inicializa el nodo con un objeto Usuario y establece el puntero siguiente a nullptr. También imprime un mensaje de depuración con el correo del usuario.

Clase ListaUsuarios

Atributos:

*Nodo *cabeza*: Un puntero que apunta al primer nodo de la lista.

*Nodo *cola*: Un puntero que apunta al último nodo de la lista.

Constructor:

ListaUsuarios(): Inicializa la lista estableciendo ambos punteros (cabeza y cola) a nullptr.

Destructor:

~*ListaUsuarios*(): Libera la memoria de todos los nodos en la lista para evitar fugas de memoria.

Métodos:

void *agregarUsuario*(const Usuario &usuario): Añade un nuevo usuario a la lista. Si el usuario ya existe (verificado por usuarioDuplicado), no lo agrega. Si la lista está vacía, el nuevo usuario se convierte en la cabeza y la cola de la lista. Si no está vacía, se añade al final y se actualiza la cola.

bool *usuarioDuplicado*(const std::string &correo) const: Verifica si un usuario con el correo dado ya existe en la lista. Si lo encuentra, retorna true; de lo contrario, false.

bool *buscarUsuarioPorCorreoyContrasena*(const std::string &correo, const std::string &contrasena) const: Busca un usuario por su correo y contraseña. Retorna true si encuentra una coincidencia; de lo contrario, false.

bool *buscarCorreo*(const std::string &correo) const: Verifica si un correo específico ya está en la lista. Retorna true si lo encuentra, false si no.

void *borrarUsuarioPorCorreo*(const std::string &correo): Elimina un usuario de la lista según su correo. Si el usuario está en la cabeza, la cabeza se actualiza. Si está en el medio o al final, se ajustan los punteros correspondientes.

void *mostrarDatosPorCorreo*(const std::string &correo) const: Muestra los datos de un usuario según su correo. Si el usuario no se encuentra, se imprime un mensaje de error.

void *generateDot*(const std::string &filename) const: Genera un archivo DOT para representar gráficamente la lista enlazada, mostrando el nombre y el correo de cada usuario. Este archivo puede ser procesado con Graphviz para visualizar la estructura de la lista.

void *renderGraphviz*(const std::string &dotFilename, const std::string &imageFilename) const: Usa Graphviz para convertir el archivo DOT generado en una imagen (formato PNG).

void *cargarUsuariosDesdeJson*(const std::string &nombreArchivo): Carga usuarios desde un archivo JSON y los agrega a la lista de usuarios.

void *registrarUsuario*(ListaUsuarios &listaUsuarios): Permite al usuario ingresar sus datos a través de la consola para registrarse. Verifica que el correo no esté duplicado antes de agregar al usuario a la lista.

Métodos auxiliares:

std::string *limpiarCadena*(const std::string &str): Limpia una cadena de caracteres eliminando espacios y caracteres innecesarios al inicio y al final.

Estos métodos permiten gestionar la lista de usuarios, realizar búsquedas, agregar o eliminar usuarios, y generar representaciones gráficas de la lista.

```cpp
class NodoMatriz
{
public:
    std::string nombreFila;
    std::string nombreColumna;
    NodoMatriz *derecha;
    NodoMatriz *abajo;

    NodoMatriz(const std::string &fila, const std::string &columna)···
};

class MatrizDispersa
{
private:
    std::vector<std::string> nombres;
    NodoMatriz *cabeza;

    // Marcar estos métodos como const
    NodoMatriz *buscarFila(const std::string &fila) const
    {
        NodoMatriz *temp = cabeza;
        while (temp && temp->nombreFila != fila)
        {
            temp = temp->abajo;
        }
        return temp;
    }

    NodoMatriz *buscarColumna(NodoMatriz *filaNodo, const std::string &columna) const
    {
        NodoMatriz *temp = filaNodo;
        while (temp && temp->nombreColumna != columna)
        {
            temp = temp->derecha;
        }
        return temp;
    }
    std::string escapeXml(const std::string &input) const
    {
        std::string escaped = input;
        std::string toReplace[][2] = {{"<", "&lt;"}, {">", "&gt;"}, {"&", "&amp;"}};
        for (const auto &pair : toReplace)
        {
            size_t pos = 0;
            while ((pos = escaped.find(pair[0], pos)) != std::string::npos)
            {
                escaped.replace(pos, pair[0].length(), pair[1]);
                pos += pair[1].length();
            }
        }
        return escaped;
    }
```

```
public:
    MatrizDispersa() : cabeza(nullptr) {}

    bool existeNombre(const std::string &nombre) const
    {
        return std::find(nombres.begin(), nombres.end(), nombre) != nombres.end();
    }

    void insertarNombre(const std::string &nombre)
    {
        if (!existeNombre(nombre))
        {
            nombres.push_back(nombre);
        }
    }
```

```cpp
void generateDotMatrizDispersa(const std::string &filename) const
{
    std::ofstream file(filename);
    if (file.is_open())
    {
        // Recolectar todos los nombres únicos
        std::unordered_set<std::string> filas;
        std::unordered_set<std::string> columnas;
        NodoMatriz *tempFila = cabeza;

        while (tempFila)
        {
            NodoMatriz *tempColumna = tempFila->derecha;
            while (tempColumna)
            {
                filas.insert(tempColumna->nombreFila);
                columnas.insert(tempColumna->nombreColumna);
                tempColumna = tempColumna->derecha;
            }
            tempFila = tempFila->abajo;
        }

        // Escribir el encabezado DOT
        file << "digraph G {" << std::endl;
        file << "node [shape=plaintext];" << std::endl;
        file << "RELACIONES_DE_AMISTAD [label=<" << std::endl;
        file << "<table border='2' cellborder='1' cellspacing='0'>" << std::endl;

        // Imprimir la primera fila con los nombres de columnas
        file << "<tr><td></td>";
        for (const auto &col : columnas)
        {
            file << "<td><b>" << escapeXml(col) << "</b></td>";
        }
        file << "</tr>" << std::endl;

        // Imprimir las filas con relaciones
        for (const auto &fila : filas)
        {
            file << "<tr><td>" << escapeXml(fila) << "</td>";

            for (const auto &col : columnas)
            {
                NodoMatriz *filaNodo = buscarFila(fila);
                if (filaNodo)
                {
                    NodoMatriz *columnaNodo = buscarColumna(filaNodo, col);
                    if (columnaNodo)
                    {
                        file << "<td>x</td>";
                    }
                    else
                    {
                        file << "<td></td>";
                    }
                }
                else
                {
                    file << "<td></td>";
                }
            }
            file << "</tr>" << std::endl;
        }

        file << "</table>>];" << std::endl;
        file << "}" << std::endl;
        file.close();
    }
    else
    {
        std::cerr << "No se pudo abrir el archivo para escribir." << std::endl;
    }
}
```

```cpp
void renderGraphvizMatrizDispersa(const std::string &dotFilename, const std::string &imageFilename) const
{
    std::string command = "dot -Tpng " + dotFilename + " -o " + imageFilename;
    int result = system(command.c_str());
    if (result != 0)
    {
        std::cerr << "Error al generar la imagen con Graphviz" << std::endl;
    }
}
```

```cpp
void generateDotMatrizDispersa_usuario(const std::string &filename, const std::string &correo) const
{
    std::ofstream file(filename);
    if (file.is_open())
    {
        // Recolectar todos los nombres únicos
        std::unordered_set<std::string> filas;
        std::unordered_set<std::string> columnas;

        NodoMatriz *tempFila = cabeza;

        while (tempFila)
        {
            NodoMatriz *tempColumna = tempFila->derecha;
            while (tempColumna)
            {
                // Solo agregar nombres si la relación involucra al usuario dado
                if (tempFila->nombreFila == correo || tempColumna->nombreColumna == correo)
                {
                    filas.insert(tempFila->nombreFila);
                    columnas.insert(tempColumna->nombreColumna);
                }
                tempColumna = tempColumna->derecha;
            }
            tempFila = tempFila->abajo;
        }

        // Escribir el encabezado DOT
        file << "digraph G {" << std::endl;
        file << "node [shape=plaintext];" << std::endl;
        file << "RELACIONES_DE_AMISTAD [label=<" << std::endl;
        file << "<table border='2' cellborder='1' cellspacing='0'>" << std::endl;

        // Imprimir la primera fila con los nombres de columnas
        file << "<tr><td></td>";
        for (const auto &col : columnas)
        {
            file << "<td><b>" << escapeXml(col) << "</b></td>";
        }
        file << "</tr>" << std::endl;

        // Imprimir las filas con relaciones
        for (const auto &fila : filas)
        {
            file << "<tr><td>" << escapeXml(fila) << "</td>";

            for (const auto &col : columnas)
            {
                NodoMatriz *filaNodo = buscarFila(fila);
                if (filaNodo)
                {
                    NodoMatriz *columnaNodo = buscarColumna(filaNodo, col);
                    if (columnaNodo && (fila == correo || col == correo))
                    {
                        file << "<td>x</td>";
                    }
                    else
                    {
                        file << "<td></td>";
                    }
                }
                else
                {
                    file << "<td></td>";
                }
            }
            file << "</tr>" << std::endl;
        }

        file << "</table>>];" << std::endl;
        file << "}" << std::endl;
        file.close();
    }
    else
    {
        std::cerr << "No se pudo abrir el archivo para escribir." << std::endl;
    }
}
```

```cpp
void renderGraphvizMatrizDispersa_usuarios(const std::string &dotFilename, const std::string &imageFilename) const
{
    std::string command = "dot -Tpng " + dotFilename + " -o " + imageFilename;
    int result = system(command.c_str());
    if (result != 0)
    {
        std::cerr << "Error al generar la imagen con Graphviz" << std::endl;
    }
}

void insertarRelacion(const std::string &nombreFila, const std::string &nombreColumna)
{
    insertarNombre(nombreFila);
    insertarNombre(nombreColumna);

    // Verificar si la fila ya existe en la matriz
    NodoMatriz *filaNodo = buscarFila(nombreFila);

    if (!filaNodo)
    {
        // Si la fila no existe, crearla
        NodoMatriz *nuevoNodoFila = new NodoMatriz(nombreFila, "");
        if (!cabeza)
        {
            cabeza = nuevoNodoFila;
        }
        else
        {
            NodoMatriz *temp = cabeza;
            while (temp->abajo)
            {
                temp = temp->abajo;
            }
            temp->abajo = nuevoNodoFila;
        }
        filaNodo = nuevoNodoFila;
    }

    // Verificar si ya existe la relación en la fila
    NodoMatriz *columnaNodo = buscarColumna(filaNodo, nombreColumna);

    if (!columnaNodo)
    {
        // Si no existe la relación, crearla y agregarla al final de la fila
        NodoMatriz *nuevoNodoColumna = new NodoMatriz(nombreFila, nombreColumna);
        NodoMatriz *temp = filaNodo;
        while (temp->derecha)
        {
            temp = temp->derecha;
        }
        temp->derecha = nuevoNodoColumna;
    }
}
```

```cpp
std::vector<std::string> obtenerAmigos(const std::string &correo) const {
    std::vector<std::string> amigos;

    // Buscar en filas
    NodoMatriz *filaNodo = buscarFila(correo);
    if (filaNodo) {
        NodoMatriz *columnaNodo = filaNodo->derecha;
        while (columnaNodo) {
            amigos.push_back(columnaNodo->nombreColumna);
            columnaNodo = columnaNodo->derecha;
        }
    }

    // Buscar en columnas
    NodoMatriz *tempFila = cabeza;
    while (tempFila) {
        NodoMatriz *tempColumna = tempFila->derecha;
        while (tempColumna) {
            if (tempColumna->nombreColumna == correo) {
                amigos.push_back(tempFila->nombreFila);
            }
            tempColumna = tempColumna->derecha;
        }
        tempFila = tempFila->abajo;
    }

    // Eliminar duplicados
    std::sort(amigos.begin(), amigos.end());
    amigos.erase(std::unique(amigos.begin(), amigos.end()), amigos.end());

    return amigos;
}

void mostrarMatriz() const
{
    NodoMatriz *tempFila = cabeza;
    std::cout << "Matriz de relaciones:" << std::endl;
    while (tempFila)
    {
        NodoMatriz *tempColumna = tempFila->derecha;
        while (tempColumna)
        {
            std::cout << "Relacion: " << tempColumna->nombreFila << " - " << tempColumna->nombreColumna << std::endl;
            tempColumna = tempColumna->derecha;
        }
        tempFila = tempFila->abajo;
    }
}

void top5ConMenosRelaciones() const
{
    std::vector<std::pair<std::string, int>> conteoRelaciones;
    NodoMatriz *tempFila = cabeza;
    while (tempFila)
    {
        NodoMatriz *tempColumna = tempFila->derecha;
        int conteo = 0;
        while (tempColumna)
        {
            conteo++;
            tempColumna = tempColumna->derecha;
        }
        conteoRelaciones.push_back({tempFila->nombreFila, conteo});
        tempFila = tempFila->abajo;
    }

    std::sort(conteoRelaciones.begin(), conteoRelaciones.end(), [](const std::pair<std::string, int> &a, const std::pair<std::string, int> &b)
              { return a.second < b.second; });

    std::cout << "Top 5 de usuarios con menos relaciones:" << std::endl;
    for (size_t i = 0; i < 5 && i < conteoRelaciones.size(); i++)
    {
        std::cout << conteoRelaciones[i].first << " - " << conteoRelaciones[i].second << " relaciones" << std::endl;
    }
}
```

```cpp
void generateDotTop5ConMenosRelaciones(const std::string &filename) const
{
    // Contar las relaciones para cada usuario en filas y columnas
    std::unordered_map<std::string, int> conteoRelaciones;
    NodoMatriz *tempFila = cabeza;

    // Contar las relaciones desde las filas
    while (tempFila)
    {
        NodoMatriz *tempColumna = tempFila->derecha;
        while (tempColumna)
        {
            conteoRelaciones[tempFila->nombreFila]++;
            conteoRelaciones[tempColumna->nombreColumna]++;
            tempColumna = tempColumna->derecha;
        }
        tempFila = tempFila->abajo;
    }

    // Convertir el conteo a un vector y ordenar por la cantidad de relaciones (ascendente)
    std::vector<std::pair<std::string, int>> conteoRelacionesVector(conteoRelaciones.begin(), conteoRelaciones.end());
    std::sort(conteoRelacionesVector.begin(), conteoRelacionesVector.end(), [](const std::pair<std::string, int> &a, const std::pair<std::string, int> &b)
              { return a.second < b.second; });

    // Seleccionar los top 5
    if (conteoRelacionesVector.size() > 5)
    {
        conteoRelacionesVector.resize(5);
    }

    // Generar el archivo DOT
    std::ofstream file(filename);
    if (file.is_open())
    {
        file << "digraph G {" << std::endl;
        file << "node [shape=plaintext];" << std::endl;
        file << "TOP5_USUARIOS [label=<" << std::endl;
        file << "<table border='2' cellborder='1' cellspacing='0'>" << std::endl;
        file << "<tr><td><b>Usuario</b></td><td><b>Relaciones</b></td></tr>" << std::endl;

        for (const auto &usuario : conteoRelacionesVector)
        {
            file << "<tr><td>" << escapeXml(usuario.first) << "</td><td>" << usuario.second << "</td></tr>" << std::endl;
        }

        file << "</table>>];" << std::endl;
        file << "}" << std::endl;
        file.close();
    }
    else
    {
        std::cerr << "No se pudo abrir el archivo para escribir." << std::endl;
    }
}

void renderGraphvizTop5(const std::string &dotFilename, const std::string &imageFilename) const
{
    std::string command = "dot -Tpng " + dotFilename + " -o " + imageFilename;
    int result = system(command.c_str());
    if (result != 0)
    {
        std::cerr << "Error al generar la imagen con Graphviz" << std::endl;
    }
}
```

```
    void eliminarRelacionesUsuario(const std::string &usuario)
    {
        NodoMatriz *tempFila = cabeza;
        while (tempFila)
        {
            NodoMatriz *tempColumna = tempFila->derecha;
            NodoMatriz *anterior = nullptr;
            while (tempColumna)
            {
                if (tempColumna->nombreFila == usuario || tempColumna->nombreColumna == usuario)
                {
                    if (anterior)
                    {
                        anterior->derecha = tempColumna->derecha;
                        delete tempColumna;
                        tempColumna = anterior->derecha;
                    }
                    else
                    {
                        tempFila->derecha = tempColumna->derecha;
                        delete tempColumna;
                        tempColumna = tempFila->derecha;
                    }
                }
                else
                {
                    anterior = tempColumna;
                    tempColumna = tempColumna->derecha;
                }
            }
            tempFila = tempFila->abajo;
        }
    }
};
```

Atributos privados:

nombres: Un vector que almacena los nombres únicos (usuarios).

cabeza: Un puntero al primer nodo de la matriz.

Métodos privados:

buscarFila: Busca una fila en la matriz a partir del nombre y devuelve el nodo correspondiente.

buscarColumna: Busca una columna específica en una fila dada.

escapeXml: Escapa caracteres especiales en cadenas para su uso en archivos DOT.

Métodos públicos:

existeNombre: Verifica si un nombre ya existe en la matriz.

insertarNombre: Añade un nuevo nombre si no existe.

generateDotMatrizDispersa: Genera un archivo DOT que representa la matriz dispersa, mostrando relaciones entre filas y columnas.

renderGraphvizMatrizDispersa: Renderiza la matriz dispersa en una imagen usando Graphviz.

generateDotMatrizDispersa_usuario: Genera un archivo DOT solo para las relaciones que involucran a un usuario específico.

insertarRelacion: Inserta una nueva relación entre dos usuarios en la matriz.

obtenerAmigos: Devuelve una lista de amigos para un usuario específico.

mostrarMatriz: Muestra las relaciones en la matriz en formato de texto.

top5ConMenosRelaciones: Muestra los cinco usuarios con menos relaciones.

generateDotTop5ConMenosRelaciones: Genera un archivo DOT para el top 5 de usuarios con menos relaciones.

Inserción de Relaciones: Las relaciones se agregan mediante insertarRelacion, creando nodos en la fila y columna correspondientes si no existen.

Visualización: Se pueden generar representaciones visuales de la matriz, ya sea completa o filtrada por usuario.

Consulta de Amistades: Con obtenerAmigos, se puede recuperar la lista de amigos de un usuario específico.

Análisis de Relaciones: Se puede analizar la estructura de la red de usuarios, identificando aquellos con menos conexiones.

Este diseño permite una gestión eficiente de una red dispersa de relaciones, y ofrece herramientas para la visualización y el análisis de la misma.

```cpp
class Relacion
{
public:
    Relacion(std::string emisor, std::string receptor, std::string estado)
        : emisor_(emisor), receptor_(receptor), estado_(estado)
    {
        std::cout << "Depuración: Relación creada con emisor: " << emisor_
                  << ", receptor: " << receptor_
                  << ", estado: " << estado_ << std::endl;
    }

    std::string getEmisor() const { return emisor_; }
    void setEmisor(const std::string &emisor)
    {
        std::cout << "Depuración: Cambiando emisor de " << emisor_ << " a " << emisor << std::endl;
        emisor_ = emisor;
    }

    std::string getReceptor() const { return receptor_; }
    void setReceptor(const std::string &receptor)
    {
        std::cout << "Depuración: Cambiando receptor de " << receptor_ << " a " << receptor << std::endl;
        receptor_ = receptor;
    }

    std::string getEstado() const { return estado_; }
    void setEstado(const std::string &estado)
    {
        std::cout << "Depuración: Cambiando estado de " << estado_ << " a " << estado << std::endl;
        estado_ = estado;
    }

private:
    std::string emisor_;
    std::string receptor_;
    std::string estado_;
};

class NodoRelacion
{
public:
    Relacion relacion;
    NodoRelacion *siguiente;

    NodoRelacion(const Relacion &relacion) : relacion(relacion), siguiente(nullptr)
    {
        std::cout << "Depuración: Nodo de relación creado para emisor: " << relacion.getEmisor()
                  << ", receptor: " << relacion.getReceptor()
                  << ", estado: " << relacion.getEstado() << std::endl;
    }
};
```

```cpp
class ListaRelaciones
{
public:
    ListaRelaciones() : cabeza(nullptr)
    {
        std::cout << "Depuración: Lista de relaciones creada." << std::endl;
    }
    NodoRelacion *obtenerCabeza() const { return cabeza; }

    ~ListaRelaciones()
    {
        NodoRelacion *actual = cabeza;
        while (actual != nullptr)
        {
            NodoRelacion *temp = actual;
            actual = actual->siguiente;
            std::cout << "Depuración: Eliminando nodo de relación con emisor: " << temp->relacion.getEmisor()
                      << ", receptor: " << temp->relacion.getReceptor() << std::endl;
            delete temp;
            temp = nullptr;
        }
    }

    void generateDotRelaciones(const std::string &filename, const std::string &usuarioEmail) const
    {
        std::ofstream file(filename);
        if (file.is_open())
        {
            file << "digraph G {" << std::endl;
            file << "node [shape=record];" << std::endl;
            file << "rankdir=LR;" << std::endl;

            NodoRelacion *current = cabeza;
            int id = 0;
            std::map<NodoRelacion *, int> nodeIds;

            // Asignar IDs a los nodos y escribir la información de los nodos en el archivo DOT
            while (current != nullptr)
            {
                if (current->relacion.getEmisor() == usuarioEmail)
                {
                    nodeIds[current] = id;

                    file << "node" << id << " [label=\"{" << "Emisor: " << current->relacion.getEmisor() << "\\n"
                         << "Receptor: " << current->relacion.getReceptor() << "\\n"
                         << "Estado: " << current->relacion.getEstado() << "}\"];" << std::endl;

                    if (current->siguiente != nullptr && current->siguiente->relacion.getEmisor() == usuarioEmail)
                    {
                        file << "node" << id << " -> node" << (id + 1) << ";" << std::endl;
                    }
                    id++;
                }
                current = current->siguiente;
            }

            file << "}" << std::endl;
            file.close();
        }
        else
        {
            std::cerr << "No se pudo abrir el archivo para escribir." << std::endl;
        }
    }

    void renderGraphvizRelaciones(const std::string &dotFilename, const std::string &imageFilename) const
    {
        std::string command = "dot -Tpng " + dotFilename + " -o " + imageFilename;
        int result = system(command.c_str());
        if (result != 0)
        {
            std::cerr << "Error al generar la imagen con Graphviz" << std::endl;
        }
    }
```

```cpp
void borrarRelacionesPorCorreo(const std::string &correo)
{
    while (cabeza && (cabeza->relacion.getEmisor() == correo || cabeza->relacion.getReceptor() == correo))
    {
        NodoRelacion *temp = cabeza;
        cabeza = cabeza->siguiente;
        delete temp;
    }

    NodoRelacion *actual = cabeza;
    while (actual && actual->siguiente)
    {
        if (actual->siguiente->relacion.getEmisor() == correo || actual->siguiente->relacion.getReceptor() == correo)
        {
            NodoRelacion *temp = actual->siguiente;
            actual->siguiente = actual->siguiente->siguiente;
            delete temp;
        }
        else
        {
            actual = actual->siguiente;
        }
    }
}

void borrarRelacionEspecifica(const std::string &correoEmisor, const std::string &correoReceptor)
{
    // Eliminar la relación específica en la cabeza de la lista
    while (cabeza && ((cabeza->relacion.getEmisor() == correoEmisor && cabeza->relacion.getReceptor() == correoReceptor) ||
                      (cabeza->relacion.getEmisor() == correoReceptor && cabeza->relacion.getReceptor() == correoEmisor)))
    {
        NodoRelacion *temp = cabeza;
        cabeza = cabeza->siguiente;
        delete temp;
    }

    // Eliminar la relación específica en el resto de la lista
    NodoRelacion *actual = cabeza;
    while (actual && actual->siguiente)
    {
        if ((actual->siguiente->relacion.getEmisor() == correoEmisor && actual->siguiente->relacion.getReceptor() == correoReceptor) ||
            (actual->siguiente->relacion.getEmisor() == correoReceptor && actual->siguiente->relacion.getReceptor() == correoEmisor))
        {
            NodoRelacion *temp = actual->siguiente;
            actual->siguiente = actual->siguiente->siguiente;
            delete temp;
        }
        else
        {
            actual = actual->siguiente;
        }
    }
}

void agregarRelacion(const Relacion &relacion)
{
    std::cout << "Depuración: Agregando relación con emisor: " << relacion.getEmisor()
              << ", receptor: " << relacion.getReceptor() << std::endl;
    NodoRelacion *nuevoNodo = new NodoRelacion(relacion);
    if (cabeza == nullptr)
    {
        cabeza = nuevoNodo;
        std::cout << "Depuración: Relación agregada como cabeza de la lista." << std::endl;
    }
    else
    {
        NodoRelacion *actual = cabeza;
        while (actual->siguiente != nullptr)
        {
            actual = actual->siguiente;
        }
        actual->siguiente = nuevoNodo;
        std::cout << "Depuración: Relación agregada al final de la lista." << std::endl;
    }
}
```

```
    void cambiarEstadoRelacion(const std::string &correoEmisor, const std::string &correoReceptor, const std::string &nuevoEstado)
    {
        NodoRelacion *actual = cabeza;
        while (actual)
        {
            // Verifica que el emisor, el receptor y el estado coincidan
            if (actual->relacion.getEmisor() == correoEmisor &&
                actual->relacion.getReceptor() == correoReceptor &&
                actual->relacion.getEstado() == "PENDIENTE")
            {
                actual->relacion.setEstado(nuevoEstado);
                return;
            }
            actual = actual->siguiente;
        }
    }

    void agregarRelacionesAceptadasAMatriz(MatrizDispersa &matriz)
    {
        NodoRelacion *actual = cabeza;

        while (actual)
        {
            if (actual->relacion.getEstado() == "ACEPTADA")
            {
                matriz.insertarRelacion(actual->relacion.getEmisor(), actual->relacion.getReceptor());
            }
            actual = actual->siguiente;
        }
    }

    void cargarRelacionesDesdeJson(const std::string &nombreArchivo)
    {
        std::ifstream archivo(nombreArchivo);
        if (archivo.is_open())
        {
            nlohmann::json jsonData;
            archivo >> jsonData;
            archivo.close();

            for (const auto &item : jsonData)
            {
                std::string emisor = item["emisor"];
                std::string receptor = item["receptor"];
                std::string estado = item["estado"];

                Relacion nuevaRelacion(emisor, receptor, estado);
                agregarRelacion(nuevaRelacion);
            }

            std::cout << "Depuración: Finalizada la carga de relaciones desde " << nombreArchivo << std::endl;
            std::cout << "-----------------------------\n";
            std::cout << "Relaciones cargadas exitosamente.\n";
            std::cout << "-----------------------------\n";
        }

        else
        {
            std::cerr << "Error al abrir el archivo JSON." << std::endl;
        }
    }

private:
    NodoRelacion *cabeza;
};
```

La clase ListaRelaciones gestiona una lista enlazada de objetos Relacion, donde cada relación representa una conexión entre un emisor y un receptor, con un estado asociado (por ejemplo, "PENDIENTE" o "ACEPTADA"). Esta clase permite agregar, eliminar, y modificar relaciones, además de ofrecer métodos para visualizar y manejar la lista de relaciones. obtenerCabeza():

Retorna el puntero al primer nodo (cabeza) de la lista de relaciones.

~*ListaRelaciones*() (Destructor):

Libera la memoria de todos los nodos de la lista de relaciones cuando la instancia de la clase se destruye.

generateDotRelaciones():

Genera un archivo DOT para representar gráficamente la lista de relaciones, enfocándose en las relaciones donde el usuario especificado es el emisor.

renderGraphvizRelaciones():

Utiliza Graphviz para generar una imagen en formato PNG a partir del archivo DOT generado.

borrarRelacionesPorCorreo():

Elimina todas las relaciones de la lista que tengan al correo especificado como emisor o receptor.

borrarRelacionEspecifica():

Elimina una relación específica de la lista donde los correos del emisor y el receptor coincidan con los proporcionados.

agregarRelacion():

Agrega una nueva relación a la lista. Si la lista está vacía, la nueva relación se convierte en la cabeza; de lo contrario, se agrega al final.

cambiarEstadoRelacion():

Cambia el estado de una relación específica de "PENDIENTE" a otro estado, basado en los correos del emisor y receptor proporcionados.

agregarRelacionesAceptadasAMatriz():

Inserta todas las relaciones con estado "ACEPTADA" en una estructura de datos externa llamada MatrizDispersa.

cargarRelacionesDesdeJson():

Carga relaciones desde un archivo JSON y las agrega a la lista de relaciones, creando nuevas instancias de Relacion para cada entrada en el archivo.

```cpp
class Emisor
{
public:
    Emisor(std::string correo, std::string receptor, std::string estado)
        : correo(correo), receptor(receptor), estado(estado)
    {
        std::cout << "Depuración: Emisor creado con correo: " << correo
                  << ", receptor: " << receptor
                  << ", estado: " << estado << std::endl;
    }
    std::string getCorreo() const { return correo; }
    std::string getReceptor() const { return receptor; }
    std::string getEstado() const { return estado; }
    void setCorreo(const std::string &nuevoCorreo)
    {
        std::cout << "Depuración: Cambiando correo de " << correo << " a " << nuevoCorreo << std::endl;
        correo = nuevoCorreo;
    }
    void setReceptor(const std::string &nuevoReceptor)
    {
        std::cout << "Depuración: Cambiando receptor de " << receptor << " a " << nuevoReceptor << std::endl;
        receptor = nuevoReceptor;
    }
    void setEstado(const std::string &nuevoEstado)
    {
        std::cout << "Depuración: Cambiando estado de " << estado << " a " << nuevoEstado << std::endl;
        estado = nuevoEstado;
    }

private:
    std::string correo;
    std::string receptor;
    std::string estado;
};

class Nodo_emisor
{
public:
    Emisor emisor;
    Nodo_emisor *siguiente;

    Nodo_emisor(const Emisor &emisor) : emisor(emisor), siguiente(nullptr) {}
};

class ListaEmisor
{
public:
    ListaEmisor() : cabeza(nullptr) {}
    Nodo_emisor *obtenerCabeza() const { return cabeza; }

    ~ListaEmisor()
    {
        while (cabeza)
        {
            Nodo_emisor *temp = cabeza;
            cabeza = cabeza->siguiente;
            delete temp;
        }
    }
```

```cpp
void generateDotEmisores(const std::string &filename) const
{
    std::ofstream file(filename);
    if (file.is_open())
    {
        file << "digraph G {" << std::endl;
        file << "node [shape=record];" << std::endl;
        file << "rankdir=LR;" << std::endl;

        Nodo_emisor *current = cabeza; // Asume que tienes una variable `cabeza` para la lista de emisores
        int id = 0;
        while (current != nullptr)
        {
            file << "node" << id << " [label=\"{" << "Correo: " << current->emisor.getCorreo() << "\\n"
                 << "Receptor: " << current->emisor.getReceptor() << "\\n"
                 << "Estado: " << current->emisor.getEstado() << "}\"];" << std::endl;
            if (current->siguiente != nullptr)
            {
                file << "node" << id << " -> node" << (id + 1) << ";" << std::endl;
            }
            current = current->siguiente;
            id++;
        }

        file << "}" << std::endl;
        file.close();
    }
    else
    {
        std::cerr << "No se pudo abrir el archivo para escribir." << std::endl;
    }
}

void renderGraphvizEmisores(const std::string &dotFilename, const std::string &imageFilename) const
{
    std::string command = "dot -Tpng " + dotFilename + " -o " + imageFilename;
    int result = system(command.c_str());
    if (result != 0)
    {
        std::cerr << "Error al generar la imagen con Graphviz" << std::endl;
    }
}
```

```cpp
void agregarEmisor(const Emisor &emisor, const ListaUsuarios &listaUsuarios, const ListaRelaciones &listaRelaciones)
{
    // Verificar si el receptor es el mismo que el emisor
    if (emisor.getCorreo() == emisor.getReceptor())
    {
        std::cout << "Error: No puedes enviarte una solicitud a ti mismo." << std::endl;
        return;
    }

    // Verificar si el receptor existe en la lista de usuarios
    if (!listaUsuarios.buscarCorreo(emisor.getReceptor()))
    {
        std::cout << "Error: El usuario con correo " << emisor.getReceptor() << " no existe en la lista de usuarios." << std::endl;
        return;
    }

    // Verificar si el emisor ya ha recibido una solicitud del receptor en la lista de relaciones
    NodoRelacion *actualRelacion2 = listaRelaciones.obtenerCabeza();
    while (actualRelacion2)
    {
        if (actualRelacion2->relacion.getEmisor() == emisor.getReceptor() &&
            actualRelacion2->relacion.getReceptor() == emisor.getCorreo() &&
            (actualRelacion2->relacion.getEstado() == "PENDIENTE" || actualRelacion2->relacion.getEstado() == "ACEPTADA"))
        {
            std::cout << "Error: Ya existe una solicitud en estado " << actualRelacion2->relacion.getEstado()
                      << " para el emisor con correo " << emisor.getCorreo() << " en la lista de relaciones." << std::endl;
            std::cout << "No se puede enviar una solicitud a un usuario que ya te ha enviado una solicitud." << std::endl;
            return;
        }
        actualRelacion2 = actualRelacion2->siguiente;
    }

    // Verificar si el emisor ya ha recibido una solicitud del receptor en la lista de emisores
    Nodo_emisor *actual2 = cabeza;
    while (actual2)
    {
        if (actual2->emisor.getCorreo() == emisor.getReceptor() &&
            actual2->emisor.getReceptor() == emisor.getCorreo() &&
            (actual2->emisor.getEstado() == "PENDIENTE" || actual2->emisor.getEstado() == "ACEPTADA"))
        {
            std::cout << "Error: Ya existe una solicitud en estado " << actual2->emisor.getEstado()
                      << " para el emisor con correo " << emisor.getCorreo() << " en la lista de emisores." << std::endl;
            std::cout << "No se puede enviar una solicitud a un usuario que ya te ha enviado una solicitud." << std::endl;
            return;
        }
        actual2 = actual2->siguiente;
    }

    // Verificar si ya existe una solicitud pendiente o aceptada para este receptor en la lista de emisores
    Nodo_emisor *actual = cabeza;
    while (actual)
    {
        if (actual->emisor.getReceptor() == emisor.getReceptor() &&
            (actual->emisor.getEstado() == "PENDIENTE" || actual->emisor.getEstado() == "ACEPTADA"))
        {
            // Condición adicional: si el emisor es el mismo, mostrar error
            if (actual->emisor.getCorreo() == emisor.getCorreo())
            {
                std::cout << "Error: Ya existe una solicitud en estado " << actual->emisor.getEstado()
                          << " para el receptor con correo " << emisor.getReceptor() << " en la lista de emisores." << std::endl;
                return;
            }
        }
        actual = actual->siguiente;
    }

    // Verificar si ya existe una solicitud pendiente o aceptada para este receptor en la lista de relaciones
    NodoRelacion *actualRelacion = listaRelaciones.obtenerCabeza();
    while (actualRelacion)
    {
        if (actualRelacion->relacion.getEmisor() == emisor.getCorreo() &&
            actualRelacion->relacion.getReceptor() == emisor.getReceptor() &&
            (actualRelacion->relacion.getEstado() == "PENDIENTE" || actualRelacion->relacion.getEstado() == "ACEPTADA"))
        {
            // Condición adicional: si el emisor es el mismo, mostrar error
            if (actualRelacion->relacion.getEmisor() == emisor.getCorreo())
            {
                std::cout << "Error: Ya existe una solicitud en estado " << actualRelacion->relacion.getEstado()
                          << " para el receptor con correo " << emisor.getReceptor() << " en la lista de relaciones." << std::endl;
                return;
            }
        }
        actualRelacion = actualRelacion->siguiente;
    }
```

```
    // Si no se encuentra ninguna solicitud pendiente o aceptada, agregar el nuevo emisor
    Nodo_emisor *nuevoNodo = new Nodo_emisor(emisor);
    nuevoNodo->siguiente = cabeza;
    cabeza = nuevoNodo;

    std::cout << "Depuración: Emisor agregado con correo: " << emisor.getCorreo() << std::endl;
}
```

```
void cambiarEstadoSolicitud(const std::string &correoEmisor, const std::string &correoReceptor, const std::string &nuevoEstado)
{
    Nodo_emisor *actual = cabeza;
    while (actual)
    {
        // Verifica que el emisor, el receptor y el estado coincidan
        if (actual->emisor.getCorreo() == correoEmisor &&
            actual->emisor.getReceptor() == correoReceptor &&
            actual->emisor.getEstado() == "PENDIENTE")
        {

            // Cambia el estado de la solicitud a nuevoEstado
            actual->emisor.setEstado(nuevoEstado);
            return;
        }
        actual = actual->siguiente;
    }
}

void borrarEmisoresPorCorreo(const std::string &correo)
{
    while (cabeza && cabeza->emisor.getCorreo() == correo)
    {
        Nodo_emisor *temp = cabeza;
        cabeza = cabeza->siguiente;
        delete temp;
    }

    Nodo_emisor *actual = cabeza;
    while (actual && actual->siguiente)
    {
        if (actual->siguiente->emisor.getCorreo() == correo)
        {
            Nodo_emisor *temp = actual->siguiente;
            actual->siguiente = actual->siguiente->siguiente;
            delete temp;
        }
        else
        {
            actual = actual->siguiente;
        }
    }
}
void borrarSolicitudEspecifica(const std::string &correoEmisor, const std::string &correoReceptor)
{
    // Eliminar la solicitud especifica en la cabeza de la lista
    while (cabeza && cabeza->emisor.getCorreo() == correoEmisor && cabeza->emisor.getReceptor() == correoReceptor)
    {
        Nodo_emisor *temp = cabeza;
        cabeza = cabeza->siguiente;
        delete temp;
    }

    // Eliminar la solicitud especifica en el resto de la lista
    Nodo_emisor *actual = cabeza;
    while (actual && actual->siguiente)
    {
        if (actual->siguiente->emisor.getCorreo() == correoEmisor &&
            actual->siguiente->emisor.getReceptor() == correoReceptor)
        {
            Nodo_emisor *temp = actual->siguiente;
            actual->siguiente = actual->siguiente->siguiente;
            delete temp;
        }
        else
        {
            actual = actual->siguiente;
        }
    }
}
```

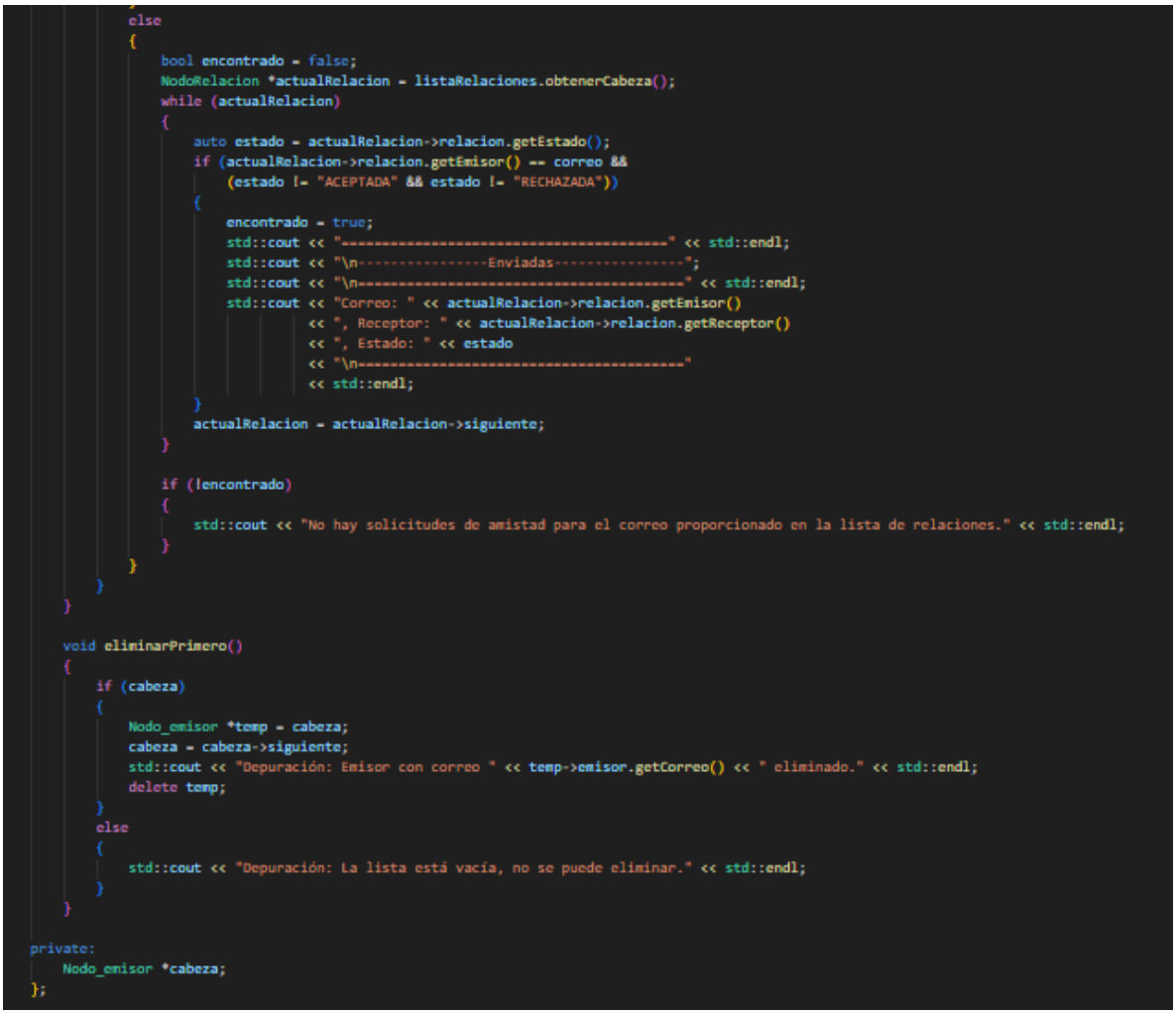
```cpp
void mostrarEmisores(const ListaRelaciones &listaRelaciones, const std::string &correo) const
{
    if (cabeza == nullptr)
    {
        // Si la lista de emisores está vacía, buscar en la lista de relaciones
        bool encontrado = false;
        NodoRelacion *actualRelacion = listaRelaciones.obtenerCabeza();
        while (actualRelacion)
        {
            auto estado = actualRelacion->relacion.getEstado();
            if (actualRelacion->relacion.getEmisor() == correo &&
                (estado != "ACEPTADA" && estado != "RECHAZADA"))
            {
                encontrado = true;
                std::cout << "=======================================" << std::endl;
                std::cout << "\n---------------Enviadas---------------";
                std::cout << "\n=======================================" << std::endl;
                std::cout << "Correo: " << actualRelacion->relacion.getEmisor()
                          << ", Receptor: " << actualRelacion->relacion.getReceptor()
                          << ", Estado: " << estado
                          << "\n======================================="
                          << std::endl;
            }
            actualRelacion = actualRelacion->siguiente;
        }

        if (!encontrado)
        {
            std::cout << "No hay solicitudes de amistad para el correo proporcionado en la lista de relaciones." << std::endl;
        }
    }
    else
    {
        // Si la lista de emisores no está vacía, mostrar y buscar en ambas listas
        Nodo_emisor *actual = cabeza;
        bool encontradoEnEmisores = false;

        while (actual)
        {
            auto estado = actual->emisor.getEstado();
            if (actual->emisor.getCorreo() == correo &&
                (estado != "ACEPTADA" && estado != "RECHAZADA"))
            {
                encontradoEnEmisores = true;
                std::cout << "=======================================" << std::endl;
                std::cout << "\n---------------Enviadas---------------";
                std::cout << "\n=======================================" << std::endl;
                std::cout << "Correo: " << actual->emisor.getCorreo()
                          << ", Receptor: " << actual->emisor.getReceptor()
                          << ", Estado: " << estado
                          << "\n======================================="
                          << std::endl;
            }
            actual = actual->siguiente;
        }

        if (encontradoEnEmisores)
        {
            // Verificar en la lista de relaciones
            bool encontradoEnRelaciones = false;
            NodoRelacion *actualRelacion = listaRelaciones.obtenerCabeza();
            while (actualRelacion)
            {
                auto estado = actualRelacion->relacion.getEstado();
                if (actualRelacion->relacion.getEmisor() == correo &&
                    (estado != "ACEPTADA" && estado != "RECHAZADA"))
                {
                    encontradoEnRelaciones = true;
                    std::cout << "=======================================" << std::endl;
                    std::cout << "\n---------------Enviadas---------------";
                    std::cout << "\n=======================================" << std::endl;
                    std::cout << "Correo: " << actualRelacion->relacion.getEmisor()
                              << ", Receptor: " << actualRelacion->relacion.getReceptor()
                              << ", Estado: " << estado
                              << "\n======================================="
                              << std::endl;
                }
                actualRelacion = actualRelacion->siguiente;
            }

            if (!encontradoEnRelaciones)
            {
                std::cout << "La solicitud no está en la lista de relaciones." << std::endl;
            }
        }
```



```
            else
            {
                bool encontrado = false;
                NodoRelacion *actualRelacion = listaRelaciones.obtenerCabeza();
                while (actualRelacion)
                {
                    auto estado = actualRelacion->relacion.getEstado();
                    if (actualRelacion->relacion.getEmisor() == correo &&
                        (estado != "ACEPTADA" && estado != "RECHAZADA"))
                    {
                        encontrado = true;
                        std::cout << "=======================================" << std::endl;
                        std::cout << "\n---------------Enviadas---------------";
                        std::cout << "\n=======================================" << std::endl;
                        std::cout << "Correo: " << actualRelacion->relacion.getEmisor()
                                  << ", Receptor: " << actualRelacion->relacion.getReceptor()
                                  << ", Estado: " << estado
                                  << "\n---------------------------------------"
                                  << std::endl;
                    }
                    actualRelacion = actualRelacion->siguiente;
                }

                if (!encontrado)
                {
                    std::cout << "No hay solicitudes de amistad para el correo proporcionado en la lista de relaciones." << std::endl;
                }
            }
        }
    }

    void eliminarPrimero()
    {
        if (cabeza)
        {
            Nodo_emisor *temp = cabeza;
            cabeza = cabeza->siguiente;
            std::cout << "Depuración: Emisor con correo " << temp->emisor.getCorreo() << " eliminado." << std::endl;
            delete temp;
        }
        else
        {
            std::cout << "Depuración: La lista está vacía, no se puede eliminar." << std::endl;
        }
    }

private:
    Nodo_emisor *cabeza;
};
```

Atributos:

correo: Cadena que representa el correo electrónico del emisor.

receptor: Cadena que representa el correo electrónico del receptor.

estado: Cadena que representa el estado de la solicitud ("PENDIENTE", "ACEPTADA", etc.).

Métodos:

Emisor(std::string correo, std::string receptor, std::string estado): Constructor que inicializa los atributos del emisor.

getCorreo(): Devuelve el correo electrónico del emisor.

getReceptor(): Devuelve el correo electrónico del receptor.

getEstado(): Devuelve el estado de la solicitud.

setCorreo(const std::string &nuevoCorreo): Cambia el correo electrónico del emisor.

setReceptor(const std::string &nuevoReceptor): Cambia el correo electrónico del receptor.

setEstado(const std::string &nuevoEstado): Cambia el estado de la solicitud.

Clase Nodo_emisor

Atributos:

emisor: Objeto de tipo Emisor que almacena los detalles del emisor.

siguiente: Puntero al siguiente nodo en la lista de emisores.

Métodos:

*Nodo*_emisor(const Emisor &emisor): Constructor que inicializa el nodo con un objeto Emisor y establece el puntero siguiente a nullptr.

Clase ListaEmisor

Atributos:

cabeza: Puntero al primer nodo de la lista de emisores.

Métodos:

~*ListaEmisor*(): Destructor que elimina todos los nodos en la lista para evitar fugas de memoria.

generateDotEmisores(const std::string &filename) const: Genera un archivo DOT que representa la lista de emisores.

renderGraphvizEmisores(const std::string &dotFilename, const std::string &imageFilename) const: Convierte el archivo DOT en una imagen PNG utilizando Graphviz.

agregarEmisor(const Emisor &emisor, const ListaUsuarios &listaUsuarios, const ListaRelaciones &listaRelaciones): Agrega un nuevo emisor a la lista, verificando varios criterios como evitar duplicados y asegurarse de que el receptor exista.

cambiarEstadoSolicitud(const std::string &correoEmisor, const std::string &correoReceptor, const std::string &nuevoEstado): Cambia el estado de una solicitud específica a un nuevo estado.

borrarEmisoresPorCorreo(const std::string &correo): Elimina todos los emisores con un correo electrónico específico.

borrarSolicitudEspecifica(const std::string &correoEmisor, const std::string &correoReceptor): Elimina una solicitud específica en función del correo del emisor y receptor.

mostrarEmisores(const ListaRelaciones &listaRelaciones, const std::string &correo) const: Muestra las solicitudes de amistad enviadas por un usuario, buscando tanto en la lista de emisores como en la lista de relaciones.

eliminarPrimero(): Elimina el primer nodo de la lista de emisores.

```cpp
class Receptor
{
public:
    Receptor(std::string correo, std::string emisor, std::string estado)
        : correo(correo), emisor(emisor), estado(estado)
    {
        std::cout << "Depuración: Receptor creado con correo: " << correo
                  << ", emisor: " << emisor
                  << ", estado: " << estado << std::endl;
    }
    std::string getCorreo() const { return correo; }
    std::string getEmisor() const { return emisor; }
    std::string getEstado() const { return estado; }
    void setCorreo(const std::string &nuevoCorreo)
    {
        std::cout << "Depuración: Cambiando correo de " << correo << " a " << nuevoCorreo << std::endl;
        correo = nuevoCorreo;
    }
    void setEmisor(const std::string &nuevoEmisor)
    {
        std::cout << "Depuración: Cambiando emisor de " << emisor << " a " << nuevoEmisor << std::endl;
        emisor = nuevoEmisor;
    }
    void setEstado(const std::string &nuevoEstado)
    {
        std::cout << "Depuración: Cambiando estado de " << estado << " a " << nuevoEstado << std::endl;
        estado = nuevoEstado;
    }

private:
    std::string correo;
    std::string emisor;
    std::string estado;
};

class Nodo_PilaReceptor
{
public:
    Receptor receptor;
    Nodo_PilaReceptor *siguiente;

    Nodo_PilaReceptor(const Receptor &receptor) : receptor(receptor), siguiente(nullptr) {}
};

class PilaReceptor
{
public:
    PilaReceptor() : cima(nullptr) {}

    ~PilaReceptor()
    {
        while (cima)
        {
            Nodo_PilaReceptor *temp = cima;
            cima = cima->siguiente;
            delete temp;
            limpiarPila();
        }
    }
    void limpiarPila()
    {
        while (!estaVacia())
        {
            desapilar(); // Usa el método desapilar para vaciar la pila
        }
        std::cout << "Depuración: La pila ha sido limpiada." << std::endl;
    }
    void apilar(const Receptor &receptor)
    {
        Nodo_PilaReceptor *nuevoNodo = new Nodo_PilaReceptor(receptor);
        nuevoNodo->siguiente = cima;
        cima = nuevoNodo;

        std::cout << "Depuración: Receptor apilado con correo: " << receptor.getCorreo() << std::endl;
    }
```

```cpp
void generateDotPilaReceptores(const std::string &filename, const std::string &usuarioEmail) const
{
    std::ofstream file(filename);
    if (file.is_open())
    {
        file << "digraph G {" << std::endl;
        file << "node [shape=record];" << std::endl;
        file << "rankdir=LR;" << std::endl;

        Nodo_PilaReceptor *current = cima; // Usamos `cima` para la pila de receptores
        int id = 0;
        std::map<Nodo_PilaReceptor *, int> nodeIds;

        // Asigna IDs a los nodos y escribe la información de los nodos en el archivo DOT
        while (current != nullptr)
        {
            if (current->receptor.getCorreo() == usuarioEmail) // Filtrar solo receptores del usuario
            {
                nodeIds[current] = id;

                file << "node" << id << " [label=\"{" << "Receptor: " << current->receptor.getCorreo() << "\\n"
                     << "Emisor: " << current->receptor.getEmisor() << "\\n"
                     << "Estado: " << current->receptor.getEstado() << "}\"];" << std::endl;

                id++;
            }
            current = current->siguiente;
        }

        current = cima;
        while (current != nullptr && current->siguiente != nullptr)
        {
            if (nodeIds.find(current) != nodeIds.end() && nodeIds.find(current->siguiente) != nodeIds.end())
            {
                int currentId = nodeIds[current];
                int siguienteId = nodeIds[current->siguiente];
                file << "node" << currentId << " -> node" << siguienteId << ";" << std::endl;
            }
            current = current->siguiente;
        }

        file << "}" << std::endl;
        file.close();
    }
    else
    {
        std::cerr << "No se pudo abrir el archivo para escribir." << std::endl;
    }
}
void renderGraphvizPilaReceptor(const std::string &dotFilename, const std::string &imageFilename) const
{
    std::string command = "dot -Tpng " + dotFilename + " -o " + imageFilename;
    int result = system(command.c_str());
    if (result != 0)
    {
        std::cerr << "Error al generar la imagen con Graphviz" << std::endl;
    }
}
```

```cpp
void borrarReceptoresPorCorreo(const std::string &correo)
{
    Nodo_PilaReceptor *nuevoCima = nullptr;

    while (cima)
    {
        if (cima->receptor.getCorreo() == correo)
        {
            Nodo_PilaReceptor *temp = cima;
            cima = cima->siguiente;
            delete temp;
        }
        else
        {
            Nodo_PilaReceptor *nodoTemporal = cima;
            cima = cima->siguiente;
            nodoTemporal->siguiente = nuevoCima;
            nuevoCima = nodoTemporal;
        }
    }

    cima = nuevoCima;
}

void buscarYApilarPendientes(const std::string &correo, const ListaEmisor &listaEmisor, const ListaRelaciones &listaRelaciones)
{
    // Buscar en ListaEmisor
    Nodo_emisor *actualEmisor = listaEmisor.obtenerCabeza();

    while (actualEmisor)
    {
        if (actualEmisor->emisor.getReceptor() == correo && actualEmisor->emisor.getEstado() == "PENDIENTE")
        {
            apilar(Receptor(correo, actualEmisor->emisor.getCorreo(), "PENDIENTE"));
        }
        actualEmisor = actualEmisor->siguiente;
    }

    // Buscar en ListaRelaciones
    NodoRelacion *actualRelacion = listaRelaciones.obtenerCabeza();

    while (actualRelacion)
    {
        if (actualRelacion->relacion.getReceptor() == correo && actualRelacion->relacion.getEstado() == "PENDIENTE")
        {
            apilar(Receptor(correo, actualRelacion->relacion.getEmisor(), "PENDIENTE"));
        }
        actualRelacion = actualRelacion->siguiente;
    }

    if (estaVacia())
    {
        std::cout << "No se encontraron solicitudes pendientes para el correo: " << correo << std::endl;
    }
    else
    {
        std::cout << "Solicitudes pendientes apiladas para el correo: " << correo << std::endl;
    }
}
```

```cpp
void rechazarSolicitud(const std::string &correoEmisor, ListaEmisor &listaEmisor, ListaRelaciones &listaRelaciones)
{
    Nodo_PilaReceptor *actual = cima;
    Nodo_PilaReceptor *anterior = nullptr;

    while (actual)
    {
        if (actual->receptor.getEmisor() == correoEmisor)
        {
            // Eliminar de la pila
            if (anterior)
            {
                anterior->siguiente = actual->siguiente;
            }
            else
            {
                cima = actual->siguiente;
            }

            // Depuración para indicar que se ha rechazado la solicitud
            std::cout << "Solicitud de amistad del emisor " << correoEmisor << " RECHAZADA." << std::endl;

            // Eliminar de la lista de emisores
            listaEmisor.borrarEmisoresPorCorreo(correoEmisor);
            listaRelaciones.borrarRelacionesPorCorreo(correoEmisor);

            delete actual;
            return;
        }
        anterior = actual;
        actual = actual->siguiente;
    }

    std::cout << "No se encontró una solicitud de amistad del emisor " << correoEmisor << " para rechazar." << std::endl;
}

void aceptarSolicitud(const std::string &correoEmisor, ListaEmisor &listaEmisor, ListaRelaciones &listaRelaciones, MatrizDispersa &matriz)
{
    Nodo_PilaReceptor *actual = cima;
    Nodo_PilaReceptor *anterior = nullptr;

    while (actual)
    {
        if (actual->receptor.getEmisor() == correoEmisor)
        {
            // Eliminar de la pila
            if (anterior)
            {
                anterior->siguiente = actual->siguiente;
            }
            else
            {
                cima = actual->siguiente;
            }

            // Depuración para indicar que se ha ACEPTADO la solicitud
            std::cout << "Solicitud de amistad del emisor " << correoEmisor << " ACEPTADA." << std::endl;

            // Obtener el nombre del receptor desde la clase relacionada
            std::string nombreReceptor = actual->receptor.getCorreo();

            // Actualizar el estado de la solicitud a ACEPTADA en la lista de emisores y relaciones
            listaEmisor.cambiarEstadoSolicitud(correoEmisor, nombreReceptor, "ACEPTADA");
            listaRelaciones.cambiarEstadoRelacion(correoEmisor, nombreReceptor, "ACEPTADA");

            // Insertar en la matriz dispersa la nueva relación de amistad
            matriz.insertarRelacion(correoEmisor, nombreReceptor);

            delete actual;
            return;
        }
        anterior = actual;
        actual = actual->siguiente;
    }

    std::cout << "No se encontró una solicitud de amistad del emisor " << correoEmisor << " para aceptar." << std::endl;
}
```

```
    void desapilar()
    {
        if (cima)
        {
            Nodo_PilaReceptor *temp = cima;
            cima = cima->siguiente;
            std::cout << "Depuración: Receptor con correo " << temp->receptor.getCorreo() << " desapilado." << std::endl;
            delete temp;
        }
        else
        {
            std::cout << "Depuración: La pila está vacía, no se puede desapilar." << std::endl;
        }
    }

    void mostrarReceptores() const
    {
        Nodo_PilaReceptor *actual = cima;
        while (actual)
        {
            std::cout << "=====================================" << std::endl;
            std::cout << "--------------Recibidas--------------";
            std::cout << "\n=====================================" << std::endl;
            ;

            std::cout << "Correo: " << actual->receptor.getCorreo()
                      << ", Emisor: " << actual->receptor.getEmisor()
                      << ", Estado: " << actual->receptor.getEstado()
                      << "\n-------------------------------------"
                      << std::endl;

            actual = actual->siguiente;
        }
    }

    bool estaVacia() const
    {
        return cima == nullptr;
    }

private:
    Nodo_PilaReceptor *cima;

    bool esDuplicado(const Receptor &receptor) const
    {
        Nodo_PilaReceptor *actual = cima;
        while (actual)
        {
            if (actual->receptor.getCorreo() == receptor.getCorreo())
            {
                return true;
            }
            actual = actual->siguiente;
        }
        return false;
    }
};
```

Atributos:

cima: Puntero al nodo superior de la pila (Nodo_PilaReceptor).

Métodos:

Destructor (~*PilaReceptor*): Elimina todos los nodos de la pila al destruir la instancia de PilaReceptor, asegurando que no queden fugas de memoria.

limpiarPila(): Elimina todos los nodos de la pila usando el método desapilar() y muestra un mensaje de depuración.

apilar(const Receptor &receptor): Agrega un nuevo receptor a la cima de la pila y muestra un mensaje de depuración.

generateDotPilaReceptores(const std::string &filename, const std::string &usuarioEmail) const: Genera un archivo DOT que visualiza la pila, filtrando solo los receptores que corresponden al correo del usuario especificado.

renderGraphvizPilaReceptor(const std::string &dotFilename, const std::string &imageFilename) const: Usa Graphviz para convertir el archivo DOT en una imagen PNG.

borrarReceptoresPorCorreo(const std::string &correo): Elimina de la pila todos los receptores con el correo especificado.

buscarYApilarPendientes(const std::string &correo, const ListaEmisor &listaEmisor, const ListaRelaciones &listaRelaciones): Busca solicitudes pendientes en ListaEmisor y ListaRelaciones, y las apila si corresponden al correo especificado.

rechazarSolicitud(const std::string &correoEmisor, ListaEmisor &listaEmisor, ListaRelaciones &listaRelaciones): Rechaza una solicitud eliminando el receptor de la pila y actualizando las listas de emisores y relaciones.

aceptarSolicitud(const std::string &correoEmisor, ListaEmisor &listaEmisor, ListaRelaciones &listaRelaciones, MatrizDispersa &matriz): Acepta una solicitud, actualiza el estado en las listas y agrega la relación a una matriz dispersa.

desapilar(): Elimina el nodo superior de la pila y muestra un mensaje de depuración.

mostrarReceptores() const: Muestra en consola todos los receptores actuales en la pila.

estaVacia() const: Verifica si la pila está vacía.

esDuplicado(const Receptor &receptor) const: Verifica si un receptor con el mismo correo ya existe en la pila.

```cpp
class Publicacion
{
public:
    Publicacion(int id, std::string correo, std::string contenido, std::string fecha, std::string hora)
        : id_(id), correo_(correo), contenido_(contenido), fecha_(fecha), hora_(hora)
    {
        std::cout << "Depuración: Publicación creada con ID: " << id_
                  << ", correo: " << correo_
                  << ", contenido: " << contenido_
                  << ", fecha: " << fecha_
                  << ", hora: " << hora_ << std::endl;
    }

    int getId() const { return id_; }
    void setID(int id)
    {
        std::cout << "Depuración: Cambiando ID de " << id_ << " a " << id << std::endl;
        id_ = id;
    }

    std::string getCorreo() const { return correo_; }
    void setCorreo(const std::string &correo)
    {
        std::cout << "Depuración: Cambiando correo de " << correo_ << " a " << correo << std::endl;
        correo_ = correo;
    }

    std::string getContenido() const { return contenido_; }
    void setContenido(const std::string &contenido)
    {
        std::cout << "Depuración: Cambiando contenido de \"" << contenido_ << "\" a \"" << contenido << "\"" << std::endl;
        contenido_ = contenido;
    }

    std::string getFecha() const { return fecha_; }
    void setFecha(const std::string &fecha)
    {
        std::cout << "Depuración: Cambiando fecha de " << fecha_ << " a " << fecha << std::endl;
        fecha_ = fecha;
    }

    std::string getHora() const { return hora_; }
    void setHora(const std::string &hora)
    {
        std::cout << "Depuración: Cambiando hora de " << hora_ << " a " << hora << std::endl;
        hora_ = hora;
    }

private:
    int id_;
    std::string correo_;
    std::string contenido_;
    std::string fecha_;
    std::string hora_;

    friend class ListaPublicaciones;
};

class NodoPublicacion
{
public:
    Publicacion publicacion;
    NodoPublicacion *siguiente;
    NodoPublicacion *anterior;

    NodoPublicacion(const Publicacion &publicacion)
        : publicacion(publicacion), siguiente(nullptr), anterior(nullptr)
    {
        std::cout << "Depuración: Nodo de publicación creado para correo: " << publicacion.getCorreo() << std::endl;
    }
};
```

```cpp
class ListaPublicaciones
{
public:
    ListaPublicaciones() : cabeza(nullptr), cola(nullptr), siguienteId(1)
    {
        std::cout << "Depuración: Lista de publicaciones creada." << std::endl;
    }

    ~ListaPublicaciones()
    {
        NodoPublicacion *actual = cabeza;
        while (actual != nullptr)
        {
            NodoPublicacion *temp = actual;
            actual = actual->siguiente;
            std::cout << "Depuración: Eliminando nodo de publicación con correo: " << temp->publicacion.getCorreo() << std::endl;
            delete temp;
        }
    }

    void generateDot(const std::string &filename) const
    {
        std::ofstream file(filename);
        if (file.is_open())
        {
            file << "digraph G {" << std::endl;
            file << "node [shape=record];" << std::endl;
            file << "rankdir=LR;" << std::endl;

            NodoPublicacion *current = cabeza;
            int id = 0;
            std::map<NodoPublicacion *, int> nodeIds;

            // Asigna IDs a los nodos y escribe la información de los nodos en el archivo DOT
            while (current != nullptr)
            {
                nodeIds[current] = id;

                file << "node" << id << " [label=\"{" << "Correo: " << current->publicacion.getCorreo() << "\\n"
                     << "Contenido: " << current->publicacion.getContenido() << "\\n"
                     << "Fecha: " << current->publicacion.getFecha() << "\\n"
                     << "Hora: " << current->publicacion.getHora() << "}\"];" << std::endl;

                current = current->siguiente;
                id++;
            }

            // Reconectar nodos y actualizar conexiones
            current = cabeza;
            while (current != nullptr)
            {
                int currentId = nodeIds[current];

                // Conectar con el siguiente nodo
                if (current->siguiente != nullptr)
                {
                    int siguienteId = nodeIds[current->siguiente];
                    file << "node" << currentId << " -> node" << siguienteId << ";" << std::endl;
                }

                // Conectar con el nodo anterior
                if (current->anterior != nullptr)
                {
                    int anteriorId = nodeIds[current->anterior];
                    if (anteriorId != currentId) // Verifica que el nodo anterior no sea el mismo que el actual
                    {
                        file << "node" << currentId << " -> node" << anteriorId << " [style=dashed];" << std::endl;
                    }
                }

                current = current->siguiente;
            }

            file << "}" << std::endl;
            file.close();
        }
```

```
    else
    {
        std::cerr << "No se pudo abrir el archivo para escribir." << std::endl;
    }
}

void generateTop5UsuariosDot(const std::string &filename) const
{
    std::ofstream file(filename);
    if (file.is_open())
    {
        // Paso 1: Contar las publicaciones por usuario
        std::map<std::string, int> publicacionesPorUsuario;
        NodoPublicacion *current = cabeza;

        while (current != nullptr)
        {
            publicacionesPorUsuario[current->publicacion.getCorreo()]++;
            current = current->siguiente;
        }

        // Paso 2: Convertir el mapa a un vector y ordenar por número de publicaciones
        std::vector<std::pair<std::string, int>> usuarios;
        for (const auto &entry : publicacionesPorUsuario)
        {
            usuarios.push_back(entry);
        }

        std::sort(usuarios.begin(), usuarios.end(),
                  [](const std::pair<std::string, int> &a, const std::pair<std::string, int> &b)
                  {
                      return b.second < a.second;
                  });

        // Paso 3: Generar la tabla en DOT
        file << "digraph G {" << std::endl;
        file << "node [shape=plaintext];" << std::endl;

        file << "Top5Usuarios [label=<" << std::endl;
        file << "<table border='1' cellborder='1' cellspacing='0'>" << std::endl;
        file << "<tr><td><b>Usuario</b></td><td><b>Publicaciones</b></td></tr>" << std::endl;

        int topN = 5;
        for (int i = 0; i < std::min(topN, (int)usuarios.size()); ++i)
        {
            file << "<tr><td>" << usuarios[i].first << "</td><td>" << usuarios[i].second << "</td></tr>" << std::endl;
        }

        file << "</table>>];" << std::endl;
        file << "}" << std::endl;

        file.close();
    }
    else
    {
        std::cerr << "No se pudo abrir el archivo para escribir." << std::endl;
    }
}
```

```cpp
void generateDotForUsuario_amigos(const std::string &usuarioCorreo, const MatrizDispersa &matriz, const std::string &filename) const
{
    std::ofstream file(filename);
    if (!file.is_open())
    {
        std::cerr << "No se pudo abrir el archivo para escribir." << std::endl;
        return;
    }

    file << "digraph G {" << std::endl;
    file << "node [shape=record];" << std::endl;
    file << "rankdir=LR;" << std::endl;

    std::vector<std::string> amigos = matriz.obtenerAmigos(usuarioCorreo);

    std::map<std::string, bool> correosAmigos;
    correosAmigos[usuarioCorreo] = true;
    for (const auto &amigo : amigos)
    {
        correosAmigos[amigo] = true;
    }

    NodoPublicacion *current = cabeza;
    int id = 0;
    std::map<NodoPublicacion *, int> nodeIds;
    NodoPublicacion *firstNode = nullptr;
    NodoPublicacion *lastNode = nullptr;

    while (current != nullptr)
    {
        if (correosAmigos.find(current->publicacion.getCorreo()) != correosAmigos.end())
        {
            nodeIds[current] = id;

            if (firstNode == nullptr) {
                firstNode = current;
            }

            lastNode = current;

            file << "node" << id << " [label=\"{" << "Correo: " << current->publicacion.getCorreo() << "\\n"
                << "Contenido: " << current->publicacion.getContenido() << "\\n"
                << "Fecha: " << current->publicacion.getFecha() << "\\n"
                << "Hora: " << current->publicacion.getHora() << "}\"];" << std::endl;

            id++;
        }
        current = current->siguiente;
    }

    for (auto it = nodeIds.begin(); it != nodeIds.end(); ++it)
    {
        NodoPublicacion *currentNode = it->first;
        int currentId = it->second;

        if (currentNode->siguiente != nullptr && nodeIds.find(currentNode->siguiente) != nodeIds.end())
        {
            int siguienteId = nodeIds[currentNode->siguiente];
            file << "node" << currentId << " -> node" << siguienteId << ";" << std::endl;

        }

        if (currentNode->anterior != nullptr && nodeIds.find(currentNode->anterior) != nodeIds.end())
        {
            int anteriorId = nodeIds[currentNode->anterior];
            if (anteriorId != currentId)
            {
                file << "node" << currentId << " -> node" << anteriorId << " [style=dashed];" << std::endl;

            }
        }
    }

    if (firstNode != nullptr && lastNode != nullptr && nodeIds.find(firstNode) != nodeIds.end() && nodeIds.find(lastNode) != nodeIds.end())
    {
        int firstId = nodeIds[firstNode];
```

```
        int firstId = nodeIds[firstNode];
        int lastId = nodeIds[lastNode];

        file << "node" << firstId << " -> node" << lastId << ";" << std::endl;
        file << "node" << lastId << " -> node" << firstId << " [style=dashed];" << std::endl;

    }
    if (nodeIds.size() > 1)
    {
        file << "node0 -> node1;" << std::endl;
    }
    if (nodeIds.size() > 1)
    {
        file << "node1 -> node0;" << std::endl;
    }

    file << "}" << std::endl;
    file.close();
}



void renderGraphviz_publis_amigos(const std::string &dotFilename, const std::string &imageFilename) const
{
    std::string command = "dot -Tpng " + dotFilename + " -o " + imageFilename;
    int result = system(command.c_str());
    if (result != 0)
    {
        std::cerr << "Error al generar la imagen con Graphviz" << std::endl;
    }
}

void renderGraphviz(const std::string &dotFilename, const std::string &imageFilename) const
{
    std::string command = "dot -Tpng " + dotFilename + " -o " + imageFilename;
    int result = system(command.c_str());
    if (result != 0)
    {
        std::cerr << "Error al generar la imagen con Graphviz" << std::endl;
    }
}

void crearPublicacion(const std::string &correo, const std::string &contenido, const std::string &fecha, const std::string &hora)
{
    if (correo.empty())
    {
        std::cerr << "Error: No se ha establecido un correo para la publicación." << std::endl;
        return;
    }

    Publicacion nuevaPublicacion(siguienteId++, correo, contenido, fecha, hora);
    agregarPublicacion(nuevaPublicacion);
}

void agregarPublicacion(const Publicacion &publicacion)
{
    std::cout << "Agregando publicación del usuario con correo: " << publicacion.getCorreo() << std::endl;

    NodoPublicacion *nuevoNodo = new NodoPublicacion(publicacion);

    if (cabeza == nullptr)
    {
        cabeza = nuevoNodo;
        cola = nuevoNodo;
        std::cout << "Publicación agregada como cabeza y cola de la lista." << std::endl;
    }
    else
    {
        cola->siguiente = nuevoNodo;
        nuevoNodo->anterior = cola;
        cola = nuevoNodo;
        std::cout << "Publicación agregada al final de la lista." << std::endl;
    }
}
```

```cpp
void borrarPublicacionesPorCorreo(const std::string &correo)
{
    while (cabeza && cabeza->publicacion.getCorreo() == correo)
    {
        NodoPublicacion *temp = cabeza;
        cabeza = cabeza->siguiente;
        delete temp;
    }

    NodoPublicacion *actual = cabeza;
    while (actual && actual->siguiente)
    {
        if (actual->siguiente->publicacion.getCorreo() == correo)
        {
            NodoPublicacion *temp = actual->siguiente;
            actual->siguiente = actual->siguiente->siguiente;
            delete temp;
        }
        else
        {
            actual = actual->siguiente;
        }
    }
}

void borrarPublicacionPorId(const std::string& correo, int id)
{
    if (cabeza == nullptr)
    {
        std::cerr << "La lista está vacía." << std::endl;
        return;
    }

    NodoPublicacion* actual = cabeza;
    NodoPublicacion* publicacionAEliminar = nullptr;

    // Buscar la publicación con el id especificado
    while (actual != nullptr)
    {
        if (actual->publicacion.getId() == id)
        {
            // Verificar si el correo coincide
            if (actual->publicacion.getCorreo() == correo)
            {
                publicacionAEliminar = actual;
                break;
            }
            else
            {
                std::cerr << "La publicación con ID: " << id << " no pertenece al correo proporcionado." << std::endl;
                return;
            }
        }
        actual = actual->siguiente;
    }

    if (publicacionAEliminar == nullptr)
    {
        std::cerr << "No se encontró publicación con ID: " << id << "." << std::endl;
        return;
    }

    if (publicacionAEliminar == cabeza)
    {
        cabeza = publicacionAEliminar->siguiente;
        if (cabeza != nullptr)
        {
            cabeza->anterior = nullptr;
        }
        else
        {
            cola = nullptr;
        }
    }
    else
    {
        publicacionAEliminar->anterior->siguiente = publicacionAEliminar->siguiente;
        if (publicacionAEliminar->siguiente != nullptr)
        {
            publicacionAEliminar->siguiente->anterior = publicacionAEliminar->anterior;
        }
        else
        {
            cola = publicacionAEliminar->anterior;
```

```
    std::cout << "Eliminando publicación con ID: " << publicacionAEliminar->publicacion.getId() << std::endl;
    delete publicacionAEliminar;
}

void mostrarTodasLasPublicaciones() const
{
    NodoPublicacion *actual = cabeza;
    if (actual == nullptr)
    {
        std::cerr << "No hay publicaciones en la lista." << std::endl;
        return;
    }

    while (actual != nullptr)
    {
        std::cout << "ID: " << actual->publicacion.getId()
                  << ", Correo: " << actual->publicacion.getCorreo()
                  << ", Contenido: " << actual->publicacion.getContenido()
                  << ", Fecha: " << actual->publicacion.getFecha()
                  << ", Hora: " << actual->publicacion.getHora() << std::endl;
        actual = actual->siguiente;
    }
}

void mostrarPublicacionesPorCorreo(const std::string &correo)
{
    NodoPublicacion *actual = cabeza;
    bool encontrado = false;

    while (actual != nullptr)
    {
        if (actual->publicacion.getCorreo() == correo)
        {
            std::cout << "ID: " << actual->publicacion.getId() << std::endl;
            std::cout << "Contenido: " << actual->publicacion.getContenido() << std::endl;
            std::cout << "Fecha: " << actual->publicacion.getFecha() << std::endl;
            std::cout << "Hora: " << actual->publicacion.getHora() << std::endl;
            std::cout << "--------------------------" << std::endl;
            encontrado = true;
        }
        actual = actual->siguiente;
    }

    if (!encontrado)
    {
        std::cout << "No se encontraron publicaciones para el usuario con correo: " << correo << std::endl;
    }
}

void mostrarPublicacionesYAmigos(const std::string &correo, const MatrizDispersa &matriz)
{
    // Muestra las publicaciones del usuario dado
    mostrarPublicacionesPorCorreo(correo);

    // Obtiene la lista de amigos del usuario
    std::vector<std::string> amigos = matriz.obtenerAmigos(correo);

    // Muestra las publicaciones de cada amigo
    for (const auto &amigo : amigos)
    {
        std::cout << "Publicaciones de " << amigo << ":" << std::endl;
        mostrarPublicacionesPorCorreo(amigo);
    }
}
```

```
    void cargarPublicacionesDesdeJson(const std::string &filename)
    {
        std::ifstream file(filename);
        if (file.is_open())
        {
            nlohmann::json jsonData;
            file >> jsonData;
            file.close();

            for (const auto &item : jsonData)
            {
                std::string correo = item["correo"];
                std::string contenido = item["contenido"];
                std::string fecha = item["fecha"];
                std::string hora = item["hora"];

                Publicacion nuevaPublicacion(siguienteId++, correo, contenido, fecha, hora);
                agregarPublicacion(nuevaPublicacion);
            }
        }
        else
        {
            std::cerr << "No se pudo abrir el archivo JSON." << std::endl;
        }
    }

private:
    NodoPublicacion *cabeza;
    NodoPublicacion *cola;
    int siguienteId;
};
```

Atributos:

*id_*: Identificador único de la publicación.

*correo_*: Correo electrónico del usuario que hizo la publicación.

*contenido_*: Texto de la publicación.

*fecha_*: Fecha en que se realizó la publicación.

*hora_*: Hora en que se realizó la publicación.

Métodos:

Constructor (Publicacion(int id, std::string correo, std::string contenido, std::string fecha, std::string hora)): Inicializa una nueva publicación con los atributos proporcionados y muestra un mensaje de depuración.

getId() const: Devuelve el ID de la publicación.

setID(int id): Cambia el ID de la publicación y muestra un mensaje de depuración.

getCorreo() const: Devuelve el correo del usuario.

setCorreo(const std::string &correo): Cambia el correo de la publicación y muestra un mensaje de depuración.

getContenido() const: Devuelve el contenido de la publicación.

setContenido(const std::string &contenido): Cambia el contenido de la publicación y muestra un mensaje de depuración.

getFecha() const: Devuelve la fecha de la publicación.

setFecha(const std::string &fecha): Cambia la fecha de la publicación y muestra un mensaje de depuración.

getHora() const: Devuelve la hora de la publicación.

setHora(const std::string &hora): Cambia la hora de la publicación y muestra un mensaje de depuración.

```cpp
int main()
{
    SetConsoleOutputCP(CP_UTF8);
    int opcion;
    ListaEmisor listaEmisor;
    PilaReceptor pilaReceptor;
    ListaUsuarios listaUsuarios;
    ListaPublicaciones listaPublicaciones;
    ListaRelaciones listaRelaciones;
    MatrizDispersa matriz;

    do
    {
        std::cout << "MENU" << std::endl;
        std::cout << "1. Iniciar sesión" << std::endl;
        std::cout << "2. Registrarse" << std::endl;
        std::cout << "3. Información" << std::endl;
        std::cout << "4. Salir" << std::endl;
        std::cout << "Ingrese su opción: ";
        std::cin >> opcion;

        // Manejo de errores de entrada
        if (std::cin.fail())
        {
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
            std::cout << "Opción inválida. Por favor, seleccione una opción válida." << std::endl;
            opcion = 0;
        }
        else
        {
            switch (opcion)
            {
            case 1:
            {
                std::cout << "Ha seleccionado la opción 1." << std::endl;

                // Solicitar correo y contraseña
                std::string correo;
                std::string contrasena;
                std::string archivo;
                std::string archivo_P;
                std::string archivo_R;
                std::string dotFilename = "usuarios.dot";
                std::string imageFilename = "usuarios.png";
                std::string dotFilenameP = "publicaciones.dot";
                std::string imageFilenameP = "publicaciones.png";
                std::string dotFilename_E = "Enviadas.dot";
                std::string imageFilename_E = "Enviadas.png";
                std::string dotFilename_R = "Enviadas_RELACIONES.dot";
                std::string imageFilename_R = "Enviadas_RELACIONES.png";
                std::string dotFilename_P = "PilaReceptor.dot";
                std::string imageFilename_P = "PilaReceptor.png";
                std::string dotFilename_Top5_MasPublicaciones = "Top5Usuarios.dot";
                std::string imageFilename_Top5_MasPublicaciones = "Top5Usuarios.png";
                std::string dotFilename_Relaciones = "relaciones.dot";
                std::string imageFilename_Relaciones = "relaciones.png";
                std::string dotFilename_top5MenosAmigos = "top5MenosAmigos.dot";
                std::string imageFilename_top5MenosAmigos = "top5MenosAmigos.png";
                std::string dotFilename_Mis_amigos = "Mis_amigos.dot";
                std::string imageFilename_Mis_amigos = "Mis_amigos.png";
                std::string dotFilename_Mis_amigos_publicaciones = "Mis_amigos_publicaciones.dot";
                std::string imageFilename_Mis_amigos_publicaciones = "Mis_amigos_publicaciones.png";
                std::string receptor;
                std::string emisor;
                std::string estado = "PENDIENTE";
                std::cout << "Ingrese su correo: ";
                std::cin >> correo;

                std::cout << "Ingrese su contraseña: ";
                std::cin >> contrasena;
```

```
if (correo == admin_correo && contrasena == admin_contrasena)
{
    std::cout << "Bienvenido, admin." << std::endl;

    int admin_opcion;
    do
    {
        std::cout << "\n----------Menu Administrador----------\n";
        std::cout << "1. Carga de usuarios\n";
        std::cout << "2. Carga de relaciones\n";
        std::cout << "3. Carga de publicaciones\n";
        std::cout << "4. Gestionar usuarios\n";
        std::cout << "5. Reportes\n";
        std::cout << "6. Regresar al menú principal\n";
        std::cout << "Seleccione una opción (0 para regresar al menú principal): ";
        std::cin >> admin_opcion;

        if (std::cin.fail())
        {
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
            std::cout << "Opción inválida. Por favor, seleccione una opción válida." << std::endl;
            admin_opcion = -1;
        }
        else
        {
            switch (admin_opcion)
            {
            case 1:
                std::cout << "Opción seleccionada: Carga de usuarios.\n";
                std::cout << "Ingrese el nombre del archivo: ";
                std::cin >> archivo;
                listaUsuarios.cargarUsuariosDesdeJson("../" + archivo + ".json");

                break;
            case 2:
                std::cout << "Opción seleccionada: Carga de relaciones.\n";
                std::cout << "Ingrese el nombre del archivo: ";
                std::cin >> archivo_R;
                listaRelaciones.cargarRelacionesDesdeJson("../" + archivo_R + ".json");
                listaRelaciones.agregarRelacionesAceptadasAMatriz(matriz);
                break;
            case 3:
                std::cout << "Opción seleccionada: Carga de publicaciones.\n";
                std::cout << "Ingrese el nombre del archivo: ";
                std::cin >> archivo_P;
                listaPublicaciones.cargarPublicacionesDesdeJson("../" + archivo_P + ".json");

                break;
            case 4:
                std::cout << "Opción seleccionada: Gestionar usuarios.\n";
                std::cout << "Ingrese el correo del usuario a borrar: ";
                std::cin >> correo;
                if (correo.empty())
                {
                    std::cerr << "No se ingresó un correo válido." << std::endl;
                }
                else
                {
                    borrarUsuario(correo, listaEmisor, pilaReceptor, listaUsuarios, listaPublicaciones, listaRelaciones, matriz);
                }

                break;
            case 5:
                int reportes_opcion;
                std::cout << "Opción seleccionada: Reportes.\n";

                do
                {
                    std::cout << "----------------Reportes----------------" << std::endl;
                    std::cout << "1. Usuarios " << std::endl;
                    std::cout << "2. Relaciones de Amistad" << std::endl;
                    std::cout << "3. Publicaciones" << std::endl;
                    std::cout << "4. Top" << std::endl;
                    std::cout << "5. Regresar al menú principal" << std::endl;
                    std::cout << "Seleccione una opción (0 para regresar al menú principal): ";
                    std::cin >> reportes_opcion;
                    switch (reportes_opcion)
                    {
                    case 1:
```

```
                        std::cout << "Opción seleccionada: Reporte de Usuarios.\n";
                        listaUsuarios.generateDot(dotFilename);
                        listaUsuarios.renderGraphviz(dotFilename, imageFilename);
                        break;
                    case 2:
                        std::cout << "Opción seleccionada: Reporte de Relaciones de Amistad.\n";
                        matriz.generateDotMatrizDispersa(dotFilename_Relaciones);
                        matriz.renderGraphvizMatrizDispersa(dotFilename_Relaciones, imageFilename_Relaciones);
                        break;
                    case 3:
                        std::cout << "Opción seleccionada: Reporte de Publicaciones.\n";
                        listaPublicaciones.generateDot(dotFilenameP);
                        listaPublicaciones.renderGraphviz(dotFilenameP, imageFilenameP);
                        break;
                    case 4:
                        std::cout << "Opción seleccionada: Reporte de Top.\n";
                        std::cout << "Generando reporte de los 5 usuarios con más publicaciones...\n";
                        listaPublicaciones.generateTop5UsuariosDot(dotFilename_Top5_MasPublicaciones);
                        listaPublicaciones.renderGraphviz(dotFilename_Top5_MasPublicaciones, imageFilename_Top5_MasPubl
                        std::cout << "Generando reporte de los 5 usuarios con menos amigos...\n";
                        matriz.generateDotTop5ConMenosRelaciones(dotFilename_top5MenosAmigos);
                        matriz.renderGraphvizTop5(dotFilename_top5MenosAmigos, imageFilename_top5MenosAmigos);
                        break;
                    case 5:
                    case 0:
                        std::cout << "Regresando al menú principal...\n";
                        reportes_opcion = 0;
                        break;
                    default:
                        break;
                    }
                } while (reportes_opcion != 0);

                break;
            case 6:
            case 0:
                std::cout << "Regresando al menú principal...\n";
                admin_opcion = 0;
                break;
            default:
                std::cout << "Opción no válida. Por favor, intente nuevamente.\n";
                break;
            }
        }
    } while (admin_opcion != 0);
}
else
{
    if (listaUsuarios.buscarUsuarioPorCorreoyContrasena(correo, contrasena))
    {
        std::cout << "Bienvenido, " << correo << std::endl;
        int usuario_opcion;
        do
        {
            std::cout << "\n----------Menu Usuario----------\n";
            std::cout << "1. Perfil\n";
            std::cout << "2. Solicitudes\n";
            std::cout << "3. Publicaciones\n";
            std::cout << "4. Reportes\n";
            std::cout << "5. Salir\n";
            std::cout << "Seleccione una opción (0 para regresar al menú principal): ";
            std::cin >> usuario_opcion;

            if (std::cin.fail())
            {
                std::cin.clear();
                std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
                std::cout << "Opción inválida. Por favor, seleccione una opción válida." << std::endl;
                usuario_opcion = -1;
            }
            else
            {
                switch (usuario_opcion)
                {
                case 1:
                    int perfil_opcion;
                    std::cout << "1. Ver perfil\n";
                    std::cout << "2. Eliminar perfil\n";
                    std::cout << "Seleccione una opción (0 para regresar al menú principal): ";
                    std::cin >> perfil_opcion;
                    if (perfil_opcion == 1)
                    {
```

```
            if (perfil_opcion == 1)
            {
                listaUsuarios.mostrarDatosPorCorreo(correo);
                break;
            }
            else if (perfil_opcion == 2)
            {
            borrarUsuario(correo, listaEmisor, pilaReceptor, listaUsuarios, listaPublicaciones, listaRelaciones, matriz);
                usuario_opcion = 0;
                break;
            }
            else
            {
                std::cout << "Opción no válida. Por favor, intente nuevamente.\n";
            }
            break;
        case 2:
            int solicitudes_opcion;
            std::cout << "----------------Solicitudes----------------\n"
                      << std::endl;
            std::cout << "1. Ver solicitudes\n";
            std::cout << "2. Enviar solicitud\n";
            std::cout << "Seleccione una opción (0 para regresar al menú principal): ";
            std::cin >> solicitudes_opcion;
            if (solicitudes_opcion == 1)
            {
                std::cout << "Solicitudes enviadas y recibidas\n";
                pilaReceptor.limpiarPila();
                pilaReceptor.buscarYApilarPendientes(correo, listaEmisor, listaRelaciones);
                listaEmisor.mostrarEmisores(listaRelaciones, correo);
                pilaReceptor.mostrarReceptores();
                std::cout << "Desea aceptar o rechazar una solicitud?\n";
                std::cout << "1. Rechazar\n";
                std::cout << "2. Aceptar\n";
                std::cout << "Seleccione una opción (0 para regresar al menú principal): ";
                int solicitudes_opcion_aceptar_rechazar;
                std::cin >> solicitudes_opcion_aceptar_rechazar;
                if (solicitudes_opcion_aceptar_rechazar == 1)
                {
                    std::cout << "Rechazar solicitud\n";
                    std::cout << "Ingrese el correo del emisor que desea rechazar: ";
                    std::cin >> emisor;
                    pilaReceptor.rechazarSolicitud(emisor, listaEmisor, listaRelaciones);
                    break;
                }
                else if (solicitudes_opcion_aceptar_rechazar == 2)
                {
                    std::cout << "Aceptar solicitud\n";
                    std::cout << "Ingrese el correo del emisor que desea aceptar: ";
                    std::cin >> emisor;
                    pilaReceptor.aceptarSolicitud(emisor, listaEmisor, listaRelaciones, matriz);
                    break;
                }
                else
                {
                    std::cout << "Opción no válida. Por favor, intente nuevamente.\n";
                }
                break;
            }
            else if (solicitudes_opcion == 2)
            {
                std::cout << "Enviar solicitud\n";
                std::cout << "Ingrese el correo del receptor: ";
                std::cin >> receptor;
                listaEmisor.agregarEmisor(Emisor(correo, receptor, estado), listaUsuarios, listaRelaciones);
                break;
            }
            else
            {
                std::cout << "Opción no válida. Por favor, intente nuevamente.\n";
            }
            break;
        case 3:
            int publicaciones_opcion;
            std::cout << "-------Publicaciones-------\n";
            std::cout << "1. Ver Todas\n";
            std::cout << "2. Crear publicación\n";
            std::cout << "3. Eliminar publicación\n";
            std::cout << "Seleccione una opción (0 para regresar al menú principal): ";
            std::cin >> publicaciones_opcion;
            if (publicaciones_opcion == 1)
            {
                std::cout << "Publicaciones\n";
                listaPublicaciones.mostrarPublicacionesYAmigos(correo, matriz);
                break;
```

```
        else if (publicaciones_opcion == 2)
        {
            std::cout << "Crear publicación\n";
            std::string contenido;
            std::string fecha;
            std::string hora;
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');

            std::cout << "Introduce el contenido de la publicación: "<< std::endl;
            std::getline(std::cin, contenido);

            std::cout << "Introduce la fecha (YYYY-MM-DD): "<< std::endl;
            std::getline(std::cin, fecha);

            std::cout << "Introduce la hora (HH:MM): "<< std::endl;
            std::getline(std::cin, hora);

            // Crear la publicación
            listaPublicaciones.crearPublicacion(correo, contenido, fecha, hora);

            std::cout << "Publicación creada exitosamente.\n";

            break;
        }
        else if (publicaciones_opcion == 3)
        {
            std::cout << "Eliminar publicación\n";
            int id;
            listaPublicaciones.mostrarPublicacionesYAmigos(correo, matriz);

            std::cout << "Ingrese el ID de la publicación que desea eliminar: ";
            std::cin >> id;
            listaPublicaciones.borrarPublicacionPorId(correo, id);

            break;
        }
        else
        {
            std::cout << "Opción no válida. Por favor, intente nuevamente.\n";
        }
        break;
    case 4:
        int reportes_opcion_usuario;
        std::cout << "Opción seleccionada: Reportes.\n";
        do
        {
            std::cout << "----------------Reportes----------------" << std::endl;
            std::cout << "1. Solicitudes Enviadas y Recibidas " << std::endl;
            std::cout << "2. Relaciones de Amistad" << std::endl;
            std::cout << "3. Publicaciones" << std::endl;
            std::cout << "4. Mis Amigos" << std::endl;
            std::cout << "5. Regresar al menú principal" << std::endl;
            std::cout << "Seleccione una opción (0 para regresar al menú principal): ";
            std::cin >> reportes_opcion_usuario;
            switch (reportes_opcion_usuario)
            {
            case 1:
                std::cout << "Opción seleccionada: Solicitudes Enviadas y Recibidas.\n";

                listaEmisor.generateDotEmisores(dotFilename_E);
                listaEmisor.renderGraphvizEmisores(dotFilename_E, imageFilename_E);
                listaRelaciones.generateDotRelaciones(dotFilename_R, correo);
                listaRelaciones.renderGraphvizRelaciones(dotFilename_R, imageFilename_R);
                pilaReceptor.generateDotPilaReceptores(dotFilename_P, correo);
                pilaReceptor.renderGraphvizPilaReceptor(dotFilename_P, imageFilename_P);
                break;
            case 2:
                std::cout << "Opción seleccionada: Reporte de Relaciones de Amistad.\n";
                matriz.generateDotMatrizDispersa(dotFilename_Relaciones);
                matriz.renderGraphvizMatrizDispersa(dotFilename_Relaciones, imageFilename_Relaciones);
                break;
            case 3:
                std::cout << "Opción seleccionada: Reporte de Publicaciones.\n";
                listaPublicaciones.generateDotForUsuario_amigos(correo, matriz, dotFilename_Mis_amigos_publicaciones);
                listaPublicaciones.renderGraphviz_publis_amigos(dotFilename_Mis_amigos_publicaciones, imageFilename_Mis_amigos_publicaciones);
                break;
            case 4:
                std::cout << "Opción seleccionada: Mis Amigos.\n";
                matriz.generateDotMatrizDispersa_usuario(dotFilename_Mis_amigos, correo);
                matriz.renderGraphvizMatrizDispersa_usuarios(dotFilename_Mis_amigos, imageFilename_Mis_amigos);
                break;
```

```
                                    case 5:
                                    case 0:
                                        std::cout << "Regresando al menú principal...\n";
                                        reportes_opcion_usuario = 0;
                                        break;
                                    default:
                                        break;
                                    }
                                } while (reportes_opcion_usuario != 0);

                                break;
                            case 5:
                            case 0:
                                std::cout << "Regresando al menú principal...\n";
                                usuario_opcion = 0;
                                break;
                            default:
                                std::cout << "Opción no válida. Por favor, intente nuevamente.\n";
                                break;
                            }
                        }
                    } while (usuario_opcion != 0);
                }
                else
                {
                    std::cout << "Correo o contraseña incorrectos." << std::endl;
                }
            }
            break;
        }
        case 2:
        {
            std::cout << "Ha seleccionado la opción REGISTRARSE." << std::endl;
            listaUsuarios.registrarUsuario(listaUsuarios);
            break;
        }

        case 3:
        {
            std::cout << "----------------Informacion del estudiante----------------" << std::endl;
            std::cout << "Nombre: Kevin Andrés Alvarez Herrera" << std::endl;
            std::cout << "Carnet: 202203038" << std::endl;
            std::cout << "Ingenieria en ciencias y sistemas" << std::endl;
            std::cout << "Curso: Estructura de datos" << std::endl;
            std::cout << "Seccion: C" << std::endl;
            std::cout << "--------------------------------" << std::endl;
            break;
        }
        case 4:
            std::cout << "Saliendo del programa..." << std::endl;
            break;
        default:
            std::cout << "Opción inválida. Por favor, seleccione una opción válida." << std::endl;
            break;
        }
    }

    std::cout << std::endl;

} while (opcion != 4);

return 0;
}
```

Inicialización:

Configura la consola para usar codificación UTF-8.

Declara variables para manejar las listas y estructuras de datos, incluyendo ListaEmisor, PilaReceptor, ListaUsuarios, ListaPublicaciones, ListaRelaciones, y MatrizDispersa.

Menú Principal:

Muestra el menú principal con opciones para iniciar sesión, registrarse, ver información, o salir.

Maneja entradas de usuario y errores de entrada.

Inicio de Sesión (Opción 1):

Solicita correo y contraseña.

Si el usuario es el administrador (correo y contraseña coinciden con los valores predefinidos), se muestra un menú de administrador con opciones para:

Cargar usuarios, relaciones, y publicaciones desde archivos JSON.

Gestionar usuarios (borrar usuarios).

Generar reportes de usuarios, relaciones de amistad, y publicaciones.

Para reportes, se generan gráficos usando Graphviz para visualizar usuarios, relaciones, publicaciones, y top 5 de usuarios con más publicaciones y menos amigos.

Menú de Usuario:

Si el usuario se autentica correctamente, se muestra un menú de usuario con opciones para:

Ver o eliminar perfil.

Gestionar solicitudes de amistad (ver, enviar, aceptar, o rechazar solicitudes).

Ver, crear, o eliminar publicaciones.

Generar reportes personalizados relacionados con solicitudes enviadas y recibidas, relaciones de amistad, publicaciones, y amigos.

Los reportes también se generan como gráficos utilizando Graphviz.

Manejo de Errores:

Se manejan entradas inválidas y se solicitan nuevamente las opciones correctas si el usuario ingresa valores no válidos.

Salir:

Permite al usuario o al administrador salir del menú y regresar al menú principal o salir del programa.

#Includes utilizados

- #include <iostream>
- #include <limits>
- #include <windows.h>
- #include <fstream>
- #include <string>
- #include <sstream>
- #include <algorithm>
- #include <cstdlib>
- #include "json.hpp"
- #include <unordered_set>

Paradigmas utilizados

POO: este se puede hacer visible gracias al uso de diferentes clases y métodos los cuales nos permitieron el fácil desarrollo y manejo de datos a la hora de programar la aplicación

Funcional: se ha utilizado en algunas partes del codigo donde se realizan operaciones sobre colecciones de datos.