Universidad de San Carlos de Guatemala

Laboratorio de Estructura de Datos

Auxiliar: Carlos Javier Castro González

Fase #1 Proyecto Socia Structure

Angel Guillermo de Jesús Pérez Jiménez

202100215

Guatemala 21 de agosto de 2024

Objetivo general

Desarrollar una aplicación de consola que simule una red social utilizando estructuras de datos no lineales, con el fin de aplicar conceptos avanzados de programación en C++ y gestionar eficientemente la información de usuarios, solicitudes de amistad y publicaciones.

Objetivos específicos

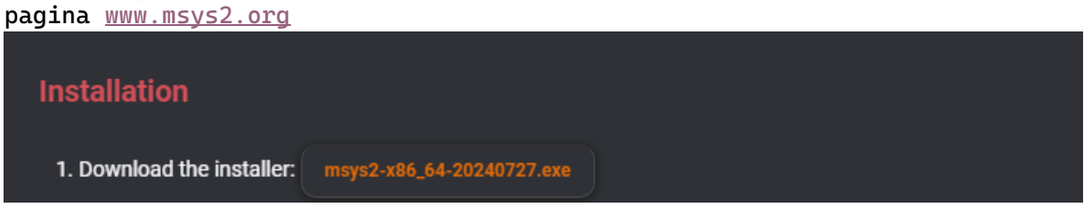
1. Implementar y gestionar las operaciones básicas de usuarios en la aplicación, incluyendo el registro, inicio de sesión, y la eliminación de cuentas, asegurando que la información de los usuarios se maneje adecuadamente en una lista simplemente enlazada.

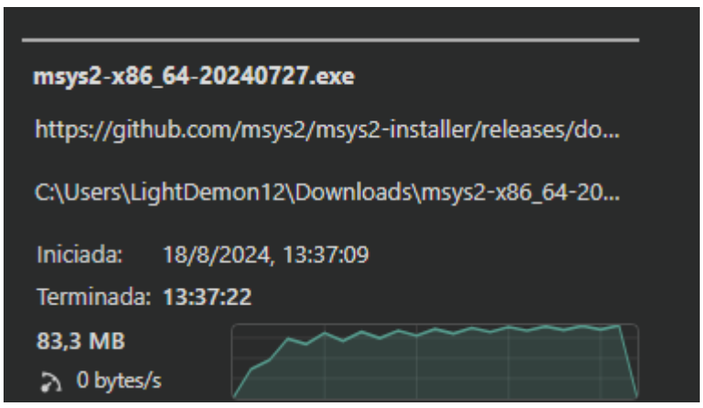
2. Desarrollar y mantener un sistema de gestión de solicitudes de amistad que permita enviar, aceptar y rechazar solicitudes, utilizando estructuras de datos como pilas y listas para asegurar una correcta administración de las solicitudes de amistad entre usuarios.

3. Diseñar y utilizar una matriz dispersa para representar y actualizar las relaciones de amistad entre usuarios, y proporcionar una funcionalidad de carga masiva de datos desde archivos JSON para facilitar la administración y análisis de la información dentro de la aplicación.

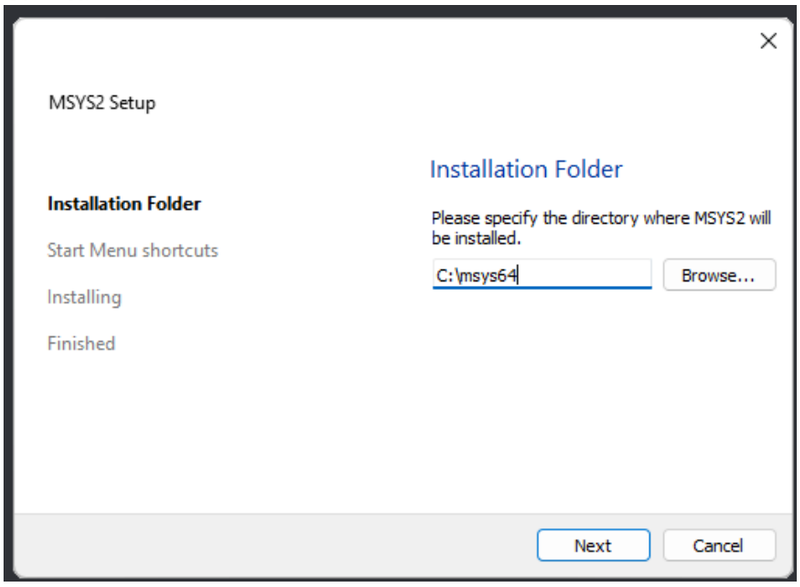
Instalación de C++ en el sistema

1. Se descargo MSYS2 para la estalación del compilador de c/c++ desde su pagina www.msys2.org

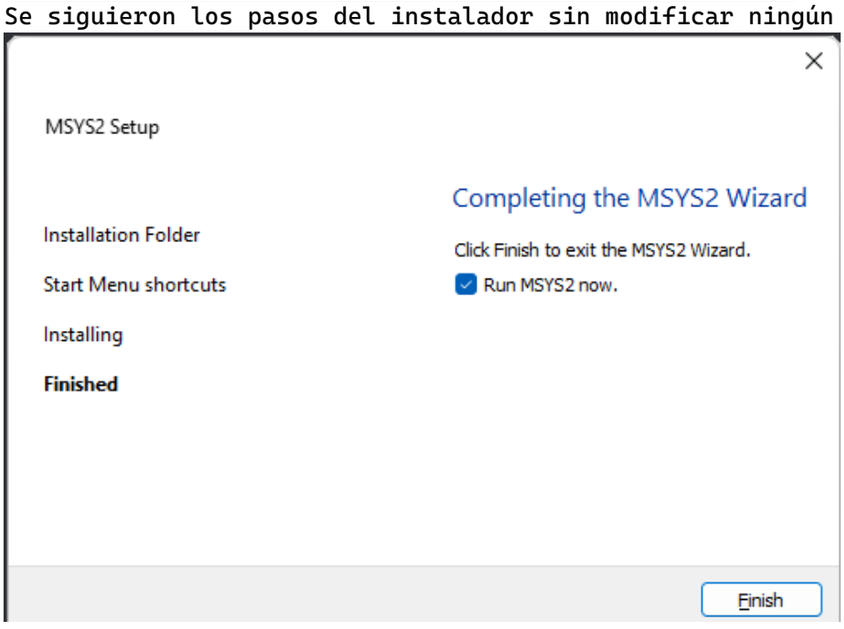




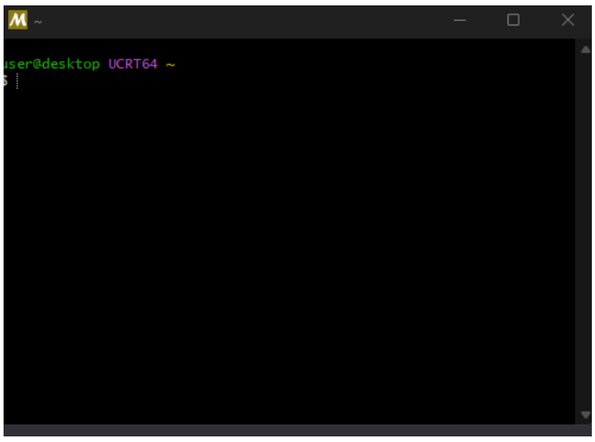
2. Una vez descargado el ejecutable se procedió a ejecutarlo para seguir con la instalación



3. Se siguieron los pasos del instalador sin modificar ningún parámetro

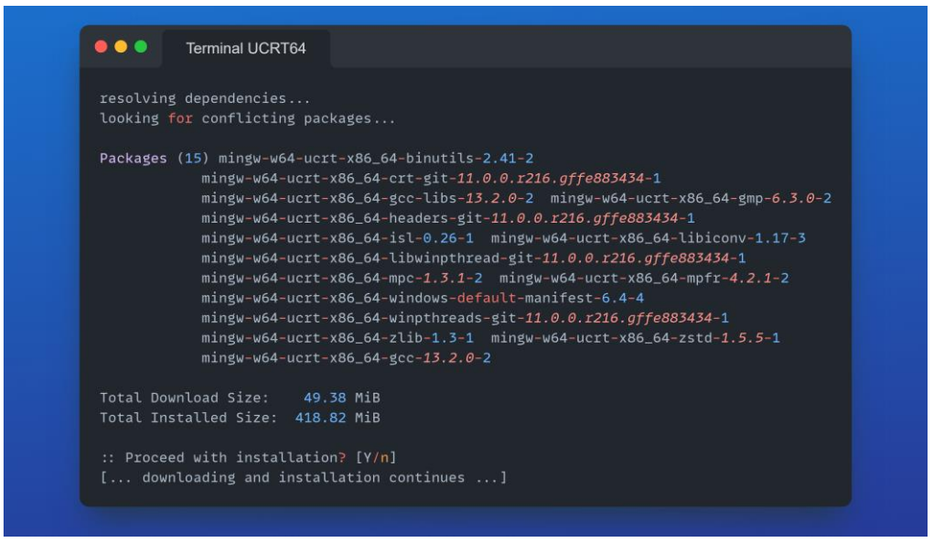


4. Una vez instalador se lanzo la terminal de UCRT74

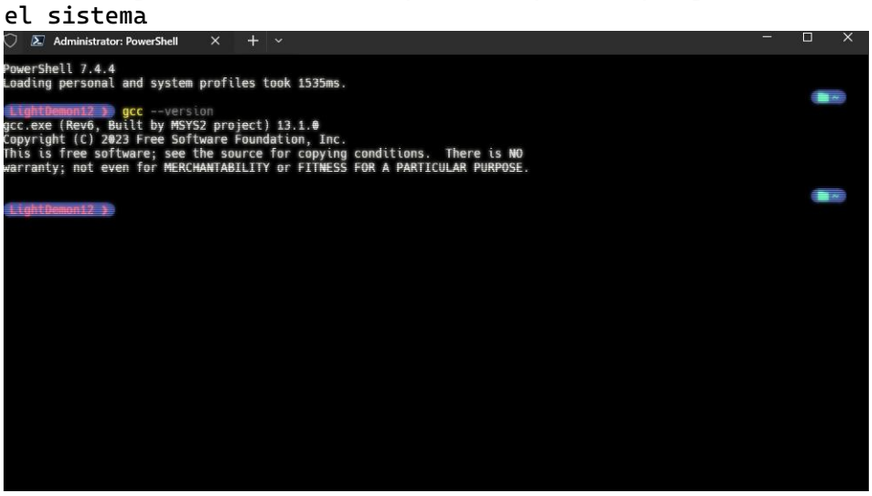


5. Dentro de esta terminal se ejecuto el siguiente código para la instalación de otras herramientas

6. La terminal muestra una salida es necesario presionar ENTER para proseguir



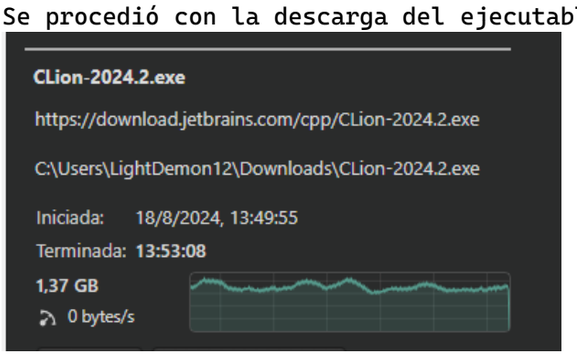
7. Con el siguiente comando se puede comprobar que gcc está instalado en el sistema



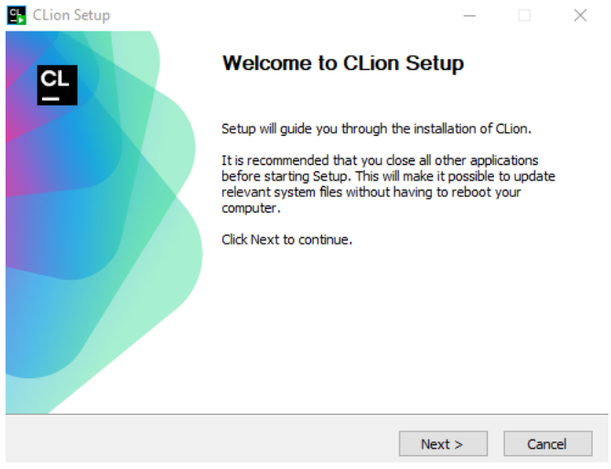
Instalación del IDE de C++ utilizado en el proyecto

1. El IDE utilizado durando el fue Clion de JETBRAINS desde su página www.jetbrains.com

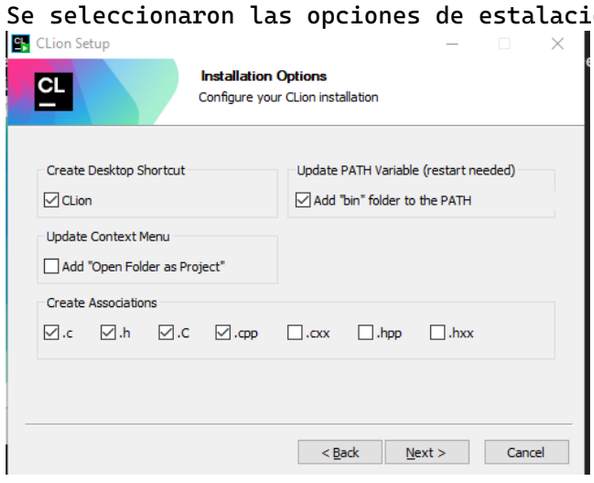
2. Se procedió con la descarga del ejecutable para su instalación



3. Se ejecuto el instalador y se siguieron los pasos sin mayores modificaciones.



4. Se seleccionaron las opciones de estalación siguientes

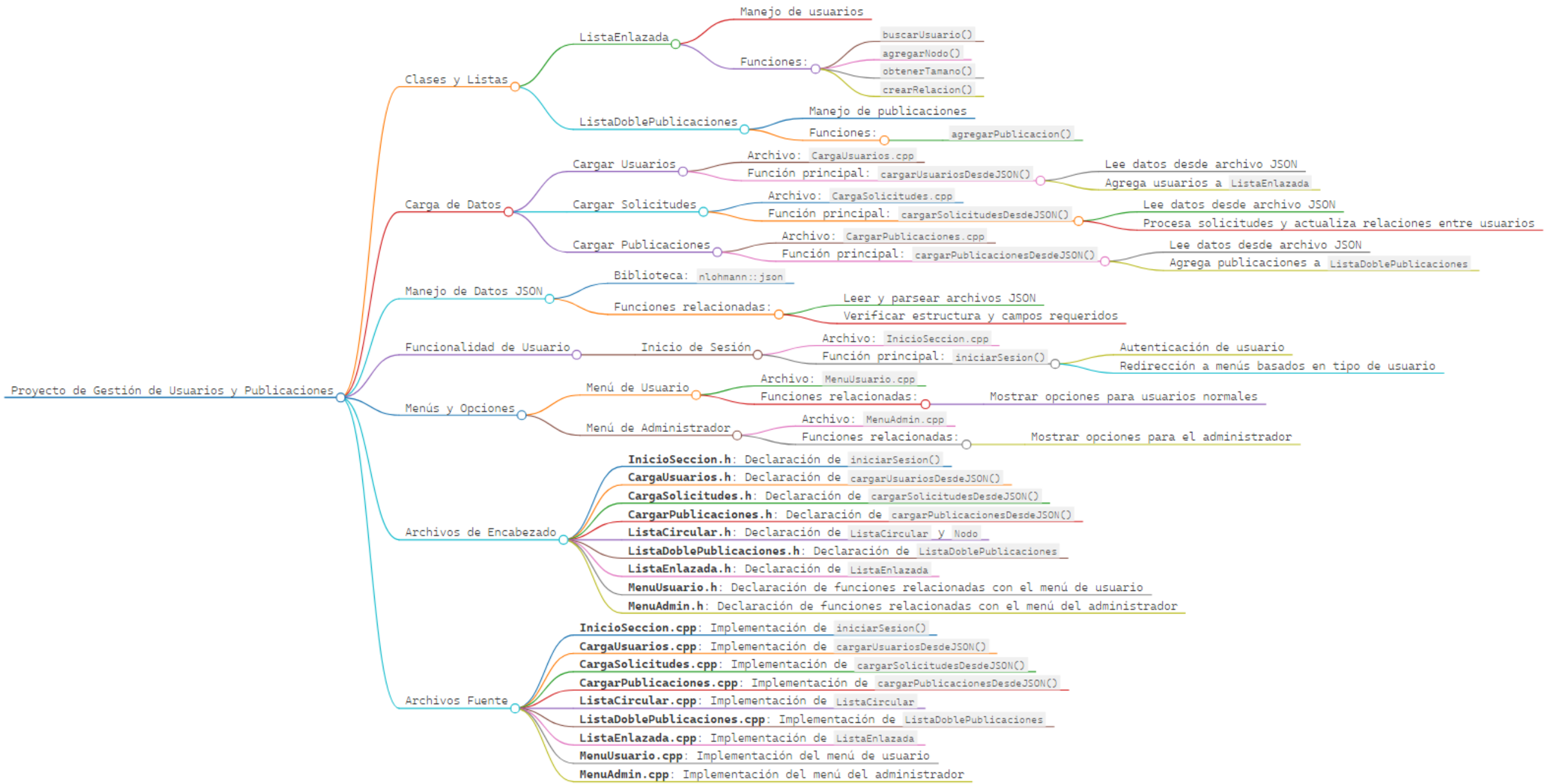


Instalación de la librería nlohmann para la lectura de archivos JSON

1. Desde el apartado de Releases en el GITHUB de nlohmann se descargo el archivo json.hpp

2. Dentro del proyecto en Clion se creo una carpeta con el nombre nlohmann donde se incluyo el archivo json.hpp descargado anteriormente.

Estructura del proyecto



Estructuras de Datos

Lista Simplemente Enlazada (ListaEnlazada):

Propósito: Gestiona la información de los usuarios registrados.

- Métodos Clave

  ✓ agregarNodo(Usuario* usuario): Agrega un nuevo usuario a la lista.

ListaEnlazada.cpp

```cpp
// Método para agregar un nuevo usuario a la lista
void ListaEnlazada::agregarNodo(Usuario* nuevoUsuario) {
    nuevoUsuario->siguiente = cabeza;
    cabeza = nuevoUsuario;
    // Insertar en la matriz dispersa
    matrizUsuarios->insertar(nuevoUsuario->correoElectronico, nuevoUsuario->nombres + " " + nuevoUsuario->apellidos);
}
```

  ✓ buscarUsuario(const std::string& correo): Busca un usuario por su correo electrónico.

ListaEnlazada.cpp

```cpp
// Método para buscar un usuario en la lista por correo electrónico y contraseña
Usuario* ListaEnlazada::buscarUsuario(const string& correoElectronico, const string& contrasena) const {
    Usuario* actual = cabeza;
    while (actual != nullptr) {
        if (actual->correoElectronico == correoElectronico && (contrasena.empty() || actual->contrasena == contrasena)) {
            return actual;
        }
        actual = actual->siguiente;
    }
    return nullptr;
}
```

  ✓ eliminarNodo(const std::string& correoElectronico): Elimina un nodo usuario por correo en varias estructuras

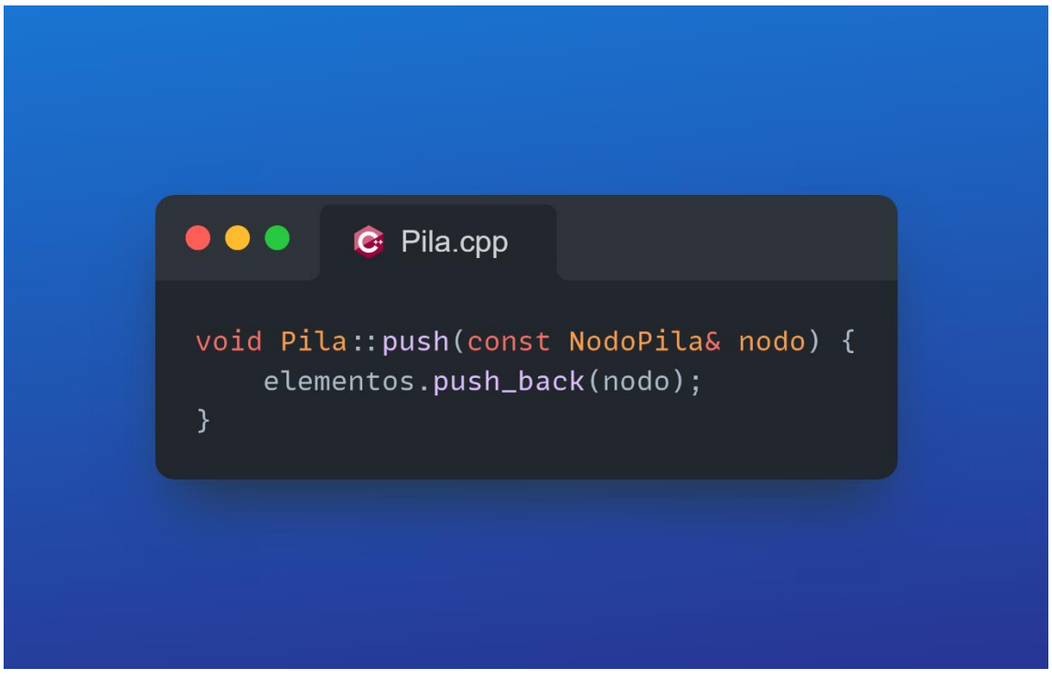
ListaEnlazada.cpp

```cpp
void ListaEnlazada::eliminarNodo(const std::string& correoElectronico) {
    Usuario* actual = cabeza;
    Usuario* anterior = nullptr;
    while (actual != nullptr && actual->correoElectronico != correoElectronico) {
        anterior = actual;
        actual = actual->siguiente;
    }
    if (actual != nullptr) {
        // Eliminar objetos de la pila asociados al usuario
        eliminarObjetosDePilaPorCorreoEmisor(correoElectronico, correoElectronico);
        // Eliminar el nodo de la lista
        if (anterior == nullptr) {
            cabeza = actual->siguiente;
        } else {
            anterior->siguiente = actual->siguiente;
        }
        // Eliminar el nodo de la matriz dispersa
        if (matrizUsuarios != nullptr) {
            matrizUsuarios->eliminarNodoPorCorreo(correoElectronico);
        } else {
            std::cerr << "Error: matrizUsuarios es nullptr." << std::endl;
        }
        delete actual;
        std::cout << "Usuario con correo " << correoElectronico << " eliminado." << std::endl;
    } else {
        std::cout << "Usuario con correo " << correoElectronico << " no encontrado." << std::endl;
    }
}
```

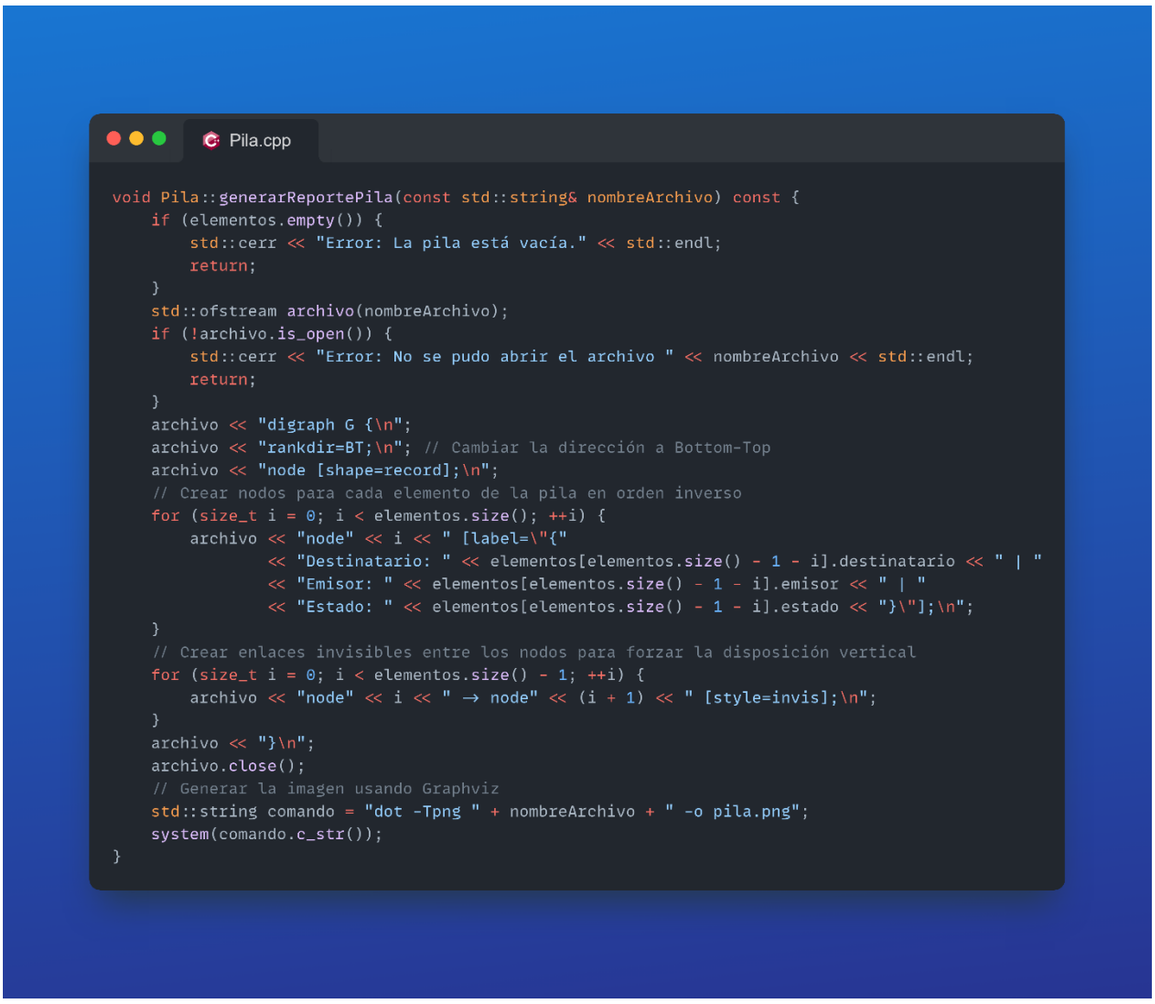
Pila (Pila)

Propósito: Almacena las solicitudes de amistad recibidas de un usuario.

- Métodos Clave:
  - ✓ push(const NodoPila& solicitud): Agrega una nueva solicitud a la pila.



  - ✓ generarReportePila(const std::string& nombreArchivo):Genera un reporte sobre los nodos de la pila



Lista Doblemente Enlazada (ListaDoblePublicaciones)

Propósito: Almacena todas las publicaciones realizadas por los usuarios.

- Métodos Clave:
  - ✓ agregarPublicacion(const std::string& correo, const std::string& contenido, const std::string& fecha, const std::string& hora):Agrega una publicación de del usuario

ListaDoblePublicaciones.cpp

```cpp
void ListaDoblePublicaciones::agregarPublicacion(const std::string& correo, const std::string& contenido, const std::string& fecha, const std::string& hora) {
    Publicacion* nuevaPublicacion = new Publicacion(correo, contenido, fecha, hora);
    if (cabeza == nullptr) {
        cabeza = cola = nuevaPublicacion;
    } else {
        cola->siguiente = nuevaPublicacion;
        nuevaPublicacion->anterior = cola;
        cola = nuevaPublicacion;
    }
}
```

Matriz Dispersa (MatrizAmistad)

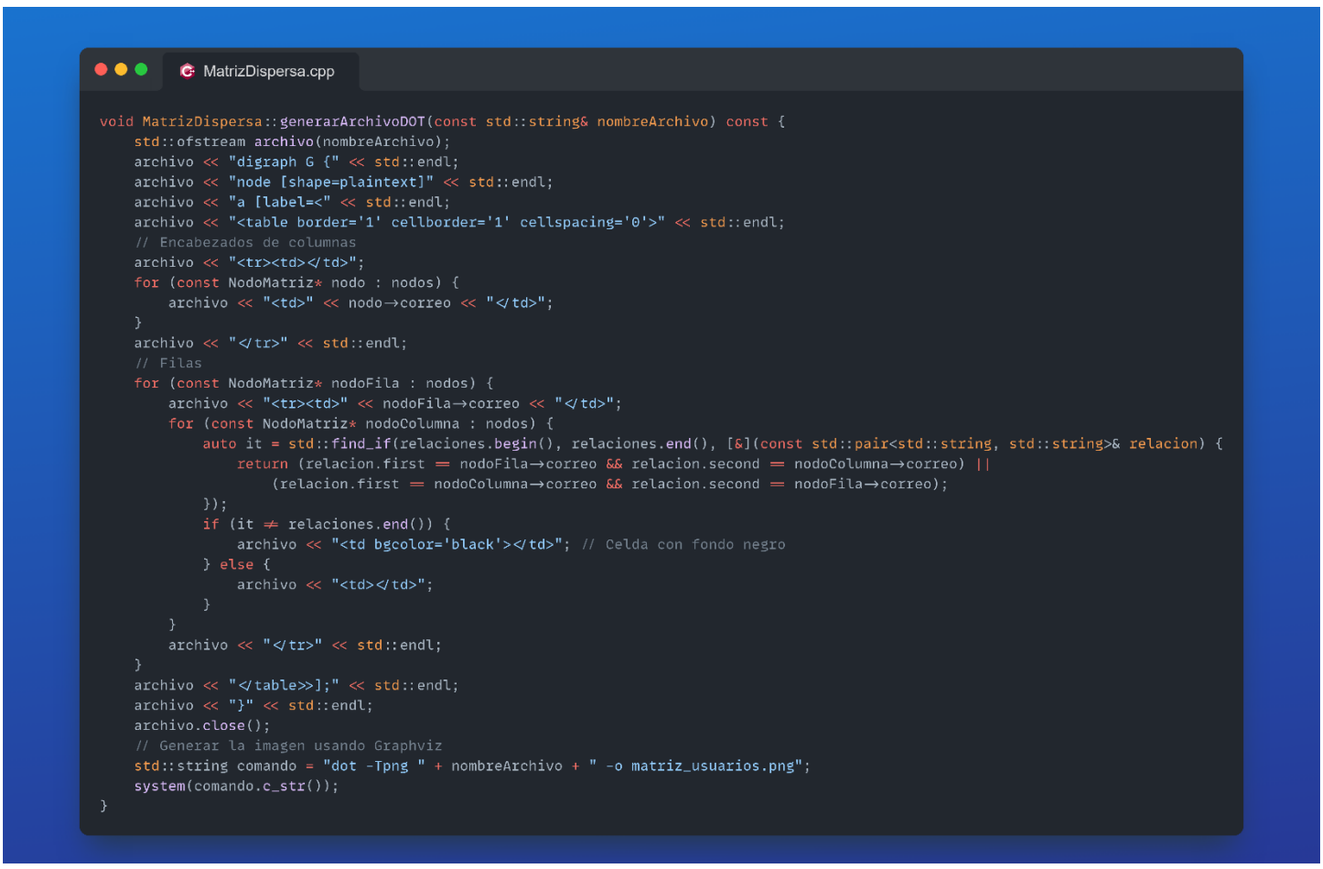
Propósito: Representa las relaciones de amistad entre usuarios.

- Métodos Clave:
  - ✓ crearRelacion(const std::string& correo1, const std::string& correo2):Crea una relación de amistad entre 2 personas

MatrizDispersa.cpp

```cpp
void MatrizDispersa::crearRelacion(const std::string& correo1, const std::string& correo2) {
    relaciones.push_back({correo1, correo2});
    std::cout << "Relación agregada a la matriz dispersa: " << correo1 << " - " << correo2 << std::endl; // Mensaje de depuración
}
```

  - ✓ MatrizDispersa::generarArchivoDOT(const std::string& nombreArchivo): Genera un reporte de las relaciones de amistad de los usuarios.



MatrizDispersa.cpp

```cpp
void MatrizDispersa::generarArchivoDOT(const std::string& nombreArchivo) const {
    std::ofstream archivo(nombreArchivo);
    archivo << "digraph G {" << std::endl;
    archivo << "node [shape=plaintext]" << std::endl;
    archivo << "a [label=<" << std::endl;
    archivo << "<table border='1' cellborder='1' cellspacing='0'>" << std::endl;
    // Encabezados de columnas
    archivo << "<tr><td></td>";
    for (const NodoMatriz* nodo : nodos) {
        archivo << "<td>" << nodo->correo << "</td>";
    }
    archivo << "</tr>" << std::endl;
    // Filas
    for (const NodoMatriz* nodoFila : nodos) {
        archivo << "<tr><td>" << nodoFila->correo << "</td>";
        for (const NodoMatriz* nodoColumna : nodos) {
            auto it = std::find_if(relaciones.begin(), relaciones.end(), [&](const std::pair<std::string, std::string>& relacion) {
                return (relacion.first == nodoFila->correo && relacion.second == nodoColumna->correo) ||
                    (relacion.first == nodoColumna->correo && relacion.second == nodoFila->correo);
            });
            if (it != relaciones.end()) {
                archivo << "<td bgcolor='black'></td>"; // Celda con fondo negro
            } else {
                archivo << "<td></td>";
            }
        }
        archivo << "</tr>" << std::endl;
    }
    archivo << "</table>>];" << std::endl;
    archivo << "}" << std::endl;
    archivo.close();
    // Generar la imagen usando Graphviz
    std::string comando = "dot -Tpng " + nombreArchivo + " -o matriz_usuarios.png";
    system(comando.c_str());
}
```