Manual De Técnico

21/08/2024

Samuel Nehemias Coyoy Perez

202200198

Contenido

Objetivos

Específicos

- Utilización de estructura de datos

Generales

- Creación de Usuarios
- Creación de solicitudes de amistad
- Creación de publicaciones

Especificación Técnica

Requisitos Hardware

- Mouse
- Teclado
- Monitor

Requisitos Software

- Sistema Operativo: Windows 10
- Herramientas: Visual Studio Code
- Lenguaje de Programación: C++

Manual Técnico: Red Social Tails

Main.cpp

Este es el archivo principal donde se carga el menú de inicio y los módulos de administrador y usuario

Se manda a llamar a la función menú principal, la cual esta tendrá las opciones de iniciar sesión, registrarse, pedir la información del estudiante o salir.

1. Menu_principal:
    a. Login: Se podrá pedir las credenciales, las cuales verificaran si es para administrador o para usuario. Para usuario se buscara en los nodos de lista simple para verificar que exista, si existe se ira a modulo usuario. A continuación se mostrará los nodos y la estructura con la cual se tiene la lista simple:

```
struct NodeListaSimple {
    string nombres;
    string apellidos;
    string fecha_de_nacimiento;
    string correo;
    string contraseña;
    NodeListaSimple* next;

    NodeListaSimple(string _nombres, string _apellidos, string
_fecha_de_nacimiento, string _correo, string _contraseña) {
        nombres = _nombres;
        apellidos = _apellidos;
        fecha_de_nacimiento = _fecha_de_nacimiento;
        correo = _correo;
        contraseña = _contraseña;
        next = nullptr;
    }
};

// Clase que define la lista simple
class ListaSimple {
private:
    NodeListaSimple* head;

public:
```

```
    ListaSimple() {
        head = nullptr;
    }

    bool correo_existente(string correo) {
        NodeListaSimple* temp = head;
        while (temp != nullptr) {
            if (temp->correo == correo) {
                return true;
            }
            temp = temp->next;
        }
        return false;
    }

    // Agregar un nodo al final de la lista
    void append(string nombres, string apellidos, string
fecha_de_nacimiento, string correo, string contraseña) {

        if(correo_existente(correo)) {
            cout << "El correo ya está registrado" << endl;
            return;
        }

        NodeListaSimple* newNode = new NodeListaSimple(nombres, apellidos,
fecha_de_nacimiento, correo, contraseña);
        if (head == nullptr) {
            head = newNode;
        } else {
            NodeListaSimple* temp = head;
            while (temp->next != nullptr) {
                temp = temp->next;
            }
            temp->next = newNode;
        }
    }

    // Agregar un nodo al inicio de la lista
    void push(string nombres, string apellidos, string fecha_de_nacimiento,
string correo, string contraseña) {
        NodeListaSimple* newNode = new NodeListaSimple(nombres, apellidos,
fecha_de_nacimiento, correo, contraseña);
        if (head == nullptr) {
            head = newNode;
        } else {
```

```
            newNode->next = head;
            head = newNode;
        }
    }

    bool verificarCredenciales(string correo, string contraseña) {
        NodeListaSimple* temp = head;
        while (temp != nullptr) {
            if (temp->correo == correo && temp->contraseña == contraseña) {
                return true;
            }
            temp = temp->next;
        }
        return false;
    }
```

b. Registrarse: Aquí podemos registrar datos como correo, nombre, apellido, contraseña, nacimiento. Por eso mismo nuestro nodo nos pide eso, porque todo usuario debe de tenerlo.

Modulo Administrador

En este modulo podremos cargar usuarios, cargar relaciones, cargar publicaciones, gestionar usuarios o ver los reportes.

- Carga Usuarios

Para cargar los usuarios se deberá seleccionar un archivo json el cual tendrá la siguiente estructura

```
    {
        "nombres":  "Juan",
        "apellidos": "Perez",
        "fecha_de_nacimiento": "1990/05/14",
        "correo": "juanperez@gmail.com",
        "contraseña" : "123456"
    }
```

Esta se guardara en la lista simple utilizando un método llamado append

```
    void append(string nombres, string apellidos, string
fecha_de_nacimiento, string correo, string contraseña) {
```

```
        if(correo_existente(correo)) {
            cout << "El correo ya está registrado" << endl;
            return;
        }

        NodeListaSimple* newNode = new NodeListaSimple(nombres, apellidos,
fecha_de_nacimiento, correo, contraseña);
        if (head == nullptr) {
            head = newNode;
        } else {
            NodeListaSimple* temp = head;
            while (temp->next != nullptr) {
                temp = temp->next;
            }
            temp->next = newNode;
        }
    }
```

Teniendo como ventaja que se a creado un método que evita repetición de correos.

- Carga relaciones

Aquí se necesitará de un json el cual necesita la siguiente estructura:

```
    {
        "emisor": "juanperez@gmail.com",
        "receptor": "mariago@gmail.com",
        "estado": "ACEPTADA"
    }
```

Donde dependiendo del estado se harán varias cosas, ejemplo: Si es PENDIENTE entonces ingresara a pila relaciones y lista amistades. Caso si es aceptada se ingresara a una matriz dispersa. A continuación mostraremos los métodos de la pila para ingresar los datos del emisor y del receptor.

```
void push(string emisor, string receptor)
{
    nodoLista *newNode = new nodoLista(emisor, receptor);
    if (top == nullptr)
    {
        top = newNode;
    }
    else
    {
        newNode->next = top;
        top = newNode;
    }
}
```

Y el método para insertar los datos a la matriz seria esta

```
void insert(string i, string j, string value = "X") {
    Node* newNode = new Node(i, j, value);

    if (!root) {
        root = new Node();
    }

    Node* row = searchRow(i);
    Node* column = searchColumn(j);

    if (!nodeExists(newNode)) {
        if (!row) {
            row = insertRowHeader(i);
        }

        if (!column) {
            column = insertColumnHeader(j);
        }

        insertInRow(newNode, row);
        insertInColumn(newNode, column);
    }
}
```

Para que funcione matriz se debe de tener todos estos métodos que explicare a continuación:

insert(string i, string j, string value = "X"): Inserta un nodo en la matriz en la posición (i, j) con un valor (por defecto "X"). Si la fila o columna no existen, las crea.

searchRow(string i): Busca y retorna el encabezado de fila con el índice i.

searchColumn(string j): Busca y retorna el encabezado de columna con el índice j.

nodeExists(Node* newNode): Verifica si ya existe un nodo en la misma posición (i, j). Si existe, actualiza el valor; si no, retorna false.

insertRowHeader(string i): Crea e inserta un nuevo encabezado de fila con el índice i.

insertColumnHeader(string j): Crea e inserta un nuevo encabezado de columna con el índice j.

insertInRow(Node* newNode, Node* rowHeader): Inserta un nodo en la fila indicada por rowHeader.

insertInColumn(Node* newNode, Node* columnHeader): Inserta un nodo en la columna indicada por columnHeader.

print() const: Imprime la matriz en la consola, incluyendo los encabezados de columnas y filas.

printColumnHeaders() const: Imprime los encabezados de las columnas en la consola.

generateDot(const string& filename) const: Genera un archivo DOT para representar la matriz y lo convierte en un archivo PNG. Utiliza el formato de tabla HTML para visualizar la matriz en el archivo DOT.

startFriendIteration(string person): Inicializa la iteración sobre los amigos de la persona dada, comenzando desde el primer amigo en la fila correspondiente.

getNextFriend(): Obtiene el siguiente amigo de la lista actual, avanzando al siguiente nodo en la fila. Retorna una cadena vacía si no hay más amigos.

- Cargar publicaciones

Para cargar las publicaciones se debe de tener un json con un formato en especifico, el cual es:

```
    {
        "correo": "juanperez@gmail.com",
        "contenido": "Si sale EDD",
```

```
        "fecha": "27/07/2024",
        "hora": "10:00"
    }
```

Al leer el json se crea un lista doblemente enlazada donde se guardan los datos utilizando este código:

```
struct node_publi {
    string correo;
    string contenido_correo;
    string fecha;
    string hora;
    node_publi* next;
    node_publi* prev;

    node_publi(string correo, string contenido_correo, string fecha, string
hora) {
        this->correo = correo;
        this->contenido_correo = contenido_correo;
        this->fecha = fecha;
        this->hora = hora;
        next = nullptr;
        prev = nullptr;
    }
};

class DoublyLinkedList {
private:
    node_publi* head;
    node_publi* tail;
public:
    DoublyLinkedList() {
        head = nullptr;
        tail = nullptr;
    }

    void append(string correo, string contenido_correo, string fecha, string
hora) {
        node_publi* newNode = new node_publi(correo, contenido_correo,
fecha, hora);
        if (head == nullptr) {
            head = newNode;
            tail = newNode;
```

```
        } else {
            tail->next = newNode;
            newNode->prev = tail;
            tail = newNode;
        }
    }

    void push(string correo, string contenido_correo, string fecha, string
hora) {
        node_publi* newNode = new node_publi(correo, contenido_correo,
fecha, hora);
        if (head == nullptr) {
            head = newNode;
            tail = newNode;
        } else {
            newNode->next = head;
            head->prev = newNode;
            head = newNode;
        }
    }
```

Si nos damos cuenta junto a ello, el nodo se encarga de pedir los datos del json que son el usuario que creo la publicación, el contenido, la hora y la fecha.

- Gestionar Usuarios

Se eliminaran usuarios de la lista simple donde fueron registrados nuestros usuarios. Esto se usaría en la lista simple para eliminar el nodo

```
    void deleteNode(string correo) {
        NodeListaSimple* temp = head;
        NodeListaSimple* prev = nullptr;

        while (temp != nullptr && temp->correo != correo) {
            prev = temp;
            temp = temp->next;
        }

        if (temp == nullptr) {
            cout << "Usuario no encontrado" << endl;
            return;
        }

        if (prev == nullptr) {
```

```
            head = temp->next;
        } else {
            prev->next = temp->next;
        }

        delete temp;
    }
```

Modulo Usuario

El modulo de usuario estará inmediatamente el login valide que los datos ingresados sean correctos para un usuario registrado.

- Perfil

Se tendrán 2 opciones, en una se podrán ver los datos del usuario actual y el otro seria de eliminar la cuenta.

Función info_usuario mostrara los datos

```
    void info_usuario(string correo) {
        NodeListaSimple* temp = head;
        while (temp != nullptr) {
            if (temp->correo == correo) {
                cout << "Nombre: " << temp->nombres << " " << temp-
>apellidos << endl;
                cout << "Fecha de Nacimiento: " << temp->fecha_de_nacimiento
<< endl;
                cout << "Correo: " << temp->correo << endl;
                cout << "Contraseña: " << temp->contraseña << endl;
                return;
            }
            temp = temp->next;
        }
        cout << "Usuario no encontrado" << endl;
    }
```

Función delete_node eliminara el nodo del usuario actual

```
void deleteNode(string correo) {
    NodeListaSimple* temp = head;
    NodeListaSimple* prev = nullptr;

    while (temp != nullptr && temp->correo != correo) {
        prev = temp;
        temp = temp->next;
    }

    if (temp == nullptr) {
        cout << "Usuario no encontrado" << endl;
        return;
    }

    if (prev == nullptr) {
        head = temp->next;
    } else {
        prev->next = temp->next;
    }

    delete temp;
}
```

- Solicitudes

Primero mostrará las solicitudes del usuario actual y luego dará las opciones de aceptar o rechazar las solicitudes. Como se había echo anteriormente con la pila y lista se borran los nodos al rechazar o aceptar. Y si se acepta se ingresa a la matriz. Estos métodos ya están especificados anteriormente.

```
matrix_amistades.insert(current->emisor, correo_usuario);
pila_relaciones.removeNode(current->emisor, correo_usuario);
listaAmistades.deleteNode(current->emisor, correo_usuario);
```

- Publicaciones

Aqui se podrá observar las publicaciones de todos los amigos, en la cual será una lista circular doblemente enlazada. Que anteriormente están guardadas en una lista doblemente enlazada.

A su vez, se podrán crear publicaciones, por eso mismo nuestro nodo nos pedirá todos los datos necesarios como, nuestro usuario, contenido, hora y fecha. Como

cada nodo ingresara un id único a cada publicación se podrán eliminar en base al id.

```
class DoublyLinkedList {
private:
    node_publi* head;
    node_publi* tail;
    int nextId; // Para asignar IDs únicos

public:
    DoublyLinkedList() : head(nullptr), tail(nullptr), nextId(0) {}

    void append(string correo, string contenido_correo, string fecha, string
hora) {
        node_publi* newNode = new node_publi(nextId++, correo,
contenido_correo, fecha, hora);
        if (head == nullptr) {
            head = newNode;
            tail = newNode;
        } else {
            tail->next = newNode;
            newNode->prev = tail;
            tail = newNode;
        }
    }

    void push(string correo, string contenido_correo, string fecha, string
hora) {
        node_publi* newNode = new node_publi(nextId++, correo,
contenido_correo, fecha, hora);
        if (head == nullptr) {
            head = newNode;
            tail = newNode;
        } else {
            newNode->next = head;
            head->prev = newNode;
            head = newNode;
        }
    }

    void print() {
        node_publi* current = head;
        while (current != nullptr) {
            cout << "ID: " << current->id << " | Correo: " << current-
>correo
```

```
                << " | Contenido: " << current->contenido_correo
                << " | Fecha: " << current->fecha << " | Hora: " <<
current->hora << " <-> ";
            current = current->next;
        }
        cout << "fin" << endl;
    }

};
```