Manual De Técnico

21/09/2024

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 y Qt creator
- Lenguaje de Programación: C++

Manual Técnico: Red Social Tails

Main.cpp

La función principal es int main(int argc, char *argv[]), donde se inicia la ejecución de la aplicación Qt. Los parámetros argc y argv permiten manejar los argumentos de la línea de comandos si es necesario.

Se crea un objeto de la clase QApplication llamado a, que es el encargado de gestionar el loop de eventos de la aplicación Qt y otros recursos globales. Este objeto es esencial para manejar las interacciones de la interfaz gráfica de usuario.

Se instancia la clase MainWindow, que es la ventana principal de la aplicación. Esta clase está definida en el archivo mainwindow.h.

La función show() se utiliza para hacer visible la ventana principal en pantalla. Sin esta llamada, la ventana no se mostraría al usuario.

La llamada a a.exec() es la que inicia el ciclo de eventos de Qt. Este ciclo se mantiene activo hasta que el usuario cierre la aplicación, permitiendo que la interfaz responda a las interacciones del usuario.

MainWindow.cpp

Este archivo contiene la implementación del constructor y destructor de la clase MainWindow, así como los métodos que gestionan los eventos de los botones de login y registro en la interfaz gráfica. En el constructor, se inicializa la interfaz llamando a ui->setupUi(this), mientras que el destructor se encarga de liberar la memoria asignada a ui. El método on_loginbtn_clicked() se activa cuando el usuario presiona el botón de login; aquí se obtienen las credenciales ingresadas y se comparan con las credenciales del administrador o las almacenadas en el árbol AVL de la aplicación. Si el login es exitoso, dependiendo de si es administrador o usuario común, se oculta la ventana de login y se muestra la ventana correspondiente. En caso de que las credenciales no sean correctas, se imprime un mensaje en la consola. Por otro lado, el método on_registrobtn_clicked() se ejecuta al presionar el botón de registro, ocultando la ventana de login y mostrando la ventana de registro si aún no ha sido creada. Este archivo es fundamental para la gestión del inicio de sesión y el flujo entre diferentes ventanas en la aplicación.

```
MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    // Inicializa la interfaz de usuario para la ventana principal.
    ui->setupUi(this);
}

MainWindow::~MainWindow()
{
    // Libera la memoria utilizada por la interfaz de usuario.
    delete ui;
}

void MainWindow::on_loginbtn_clicked()
{
    // Definición de credenciales de administrador.
    std::string userAd = "admin@gmail.com";
    std::string passAd = "EDD2S2024";

    // Obtención de las credenciales ingresadas por el usuario.
    QString usuario = ui->userline->text();
    QString password = ui->passline->text();

    // Verificación de administrador.
    if(userAd == usuario.toStdString() && passAd == password.toStdString()) {
        if(!ventanaAdmin) {
            ventanaAdmin = new admin(this);
```

```
        }
        this->hide(); // Oculta la ventana de login.
        ventanaAdmin->show(); // Muestra la ventana de administración.
    }
    // Verificación de usuario común en el árbol AVL.
    else
if(AppData::getInstance().getAVLTree().verifyCredentials(usuario.toStdString(),
password.toStdString())) {
        if(!ventanaUser) {
            ventanaUser = new user(this);
        }
        this->hide(); // Oculta la ventana de login.
        ventanaUser->show(); // Muestra la ventana de usuario.
    }
    // Verificación de credenciales de usuario "user".
    else if(usuario.toStdString() == "user" && password.toStdString() == "user") {
        if(!ventanaUser) {
            ventanaUser = new user(this);
        }
        this->hide(); // Oculta la ventana de login.
        ventanaUser->show(); // Muestra la ventana de usuario.
    }
    else {
        // Muestra un mensaje de error si las credenciales son incorrectas.
        qDebug() << "Usuario o contraseña incorrecta";
    }
}


void MainWindow::on_registrobtn_clicked()
```

```
{
    // Muestra la ventana de registro y oculta la ventana de login.
    if(!ventanaRegistro) {
        ventanaRegistro = new registro(this);
    }
    this->hide(); // Oculta la ventana de login.
    ventanaRegistro->show(); // Muestra la ventana de registro.
}
```

Admin.cpp

Descripción General

El archivo `admin.cpp` define la clase `admin`, que es una ventana de diálogo en la aplicación de administración. Esta clase permite gestionar usuarios mediante una interfaz gráfica, incluyendo la carga de datos, visualización, modificación y eliminación de usuarios.

Funcionalidades Clave

1. **Carga de Usuarios desde JSON (`on_cargarUserbtn_clicked`)**
   - Permite seleccionar un archivo JSON que contiene datos de usuarios.
   - Los datos se leen y se insertan en el árbol AVL global gestionado por `AppData`.

2. **Visualización de Usuarios (`on_showUserbtn_clicked`)**
   - Realiza un recorrido en orden del árbol AVL y muestra los datos en una tabla.

3. **Mostrar Datos en Tabla (`mostrarDatosEnTabla`)**
   - Limpia la tabla y llena filas con los datos de los usuarios del árbol AVL.

- Agrega botones para modificar y eliminar usuarios, con funcionalidades asociadas.

4. **Eliminar Usuario (`eliminarUsuario`)**

- Elimina un usuario del árbol AVL y actualiza la tabla.

5. **Buscar Usuario (`on_buscarbtn_clicked`)**

- Busca un usuario en el árbol AVL basado en un correo electrónico ingresado.

- Muestra los datos del usuario encontrado en la tabla.

6. **Recorridos de Árbol (`on_ordenbtn_clicked`)**

- Permite seleccionar diferentes métodos de recorrido (InOrder, PreOrder, PostOrder) para visualizar usuarios en la tabla.

Detalles de Implementación

- **Interacción con el Diálogo de Modificación**

- Se oculta la ventana principal cuando se abre el diálogo para modificar un usuario.

- Se actualizan los datos en el árbol AVL y en la tabla después de la modificación.

- **Conexión de Señales y Slots**

- Se utilizan `QPushButton` para modificar y eliminar usuarios. Las señales de clic se conectan a las respectivas funciones de modificación y eliminación.

```
admin::admin(QWidget *parent)
    : QDialog(parent)
    , ui(new Ui::admin)
{
    ui->setupUi(this);
```

```
    avlTemporal = new AVL();  // Inicializa un AVL temporal
}

admin::~admin()
{
    delete ui;
    delete avlTemporal;  // Libera la memoria del AVL temporal
}

void admin::on_cargarUserbtn_clicked()
{
    // Cargar usuarios desde un archivo JSON
    AppData& appData = AppData::getInstance();
    QString ruta = QFileDialog::getOpenFileName(this, "Open file", "/", "Text Files
(*.json);;All Files (*,*)");

    if (!ruta.isEmpty()) {
        QFile file(ruta);
        if (file.open(QFile::ReadOnly)) {
            QByteArray jsonData = file.readAll();
            file.close();
            QJsonDocument doc = QJsonDocument::fromJson(jsonData);

            if (doc.isArray()) {
                QJsonArray root = doc.array();
                foreach (const QJsonValue &v, root) {
                    QJsonObject obj = v.toObject();
                    // Extrae los datos del JSON y los inserta en el AVL
                    QString correo = obj.value("correo").toString();
```

```
                appData.getAVLTree().insert(correo.toStdString(), /* otros datos */);
            }
            QMessageBox::information(this, "Éxito", "Los datos se han cargado exitosamente.");
        } else {
            QMessageBox::warning(this, "Error", "El archivo JSON no es un array.");
        }
    } else {
        QMessageBox::warning(this, "Error", "No se pudo abrir el archivo.");
    }
}

void admin::on_cargarSolibtn_clicked()
{
    // Función no implementada
}

void admin::on_cargarPublibtn_clicked()
{
    // Función no implementada
}

void admin::on_pushButton_6_clicked()
{
    // Cerrar sesión y mostrar la ventana principal
    MainWindow *ventana = new MainWindow(this);
    this->close();
    ventana->show();
```

```
}

void admin::on_showUserbtn_clicked()
{
    // Mostrar usuarios en consola usando el recorrido in-order del AVL
    AppData& appData = AppData::getInstance();
    appData.getAVLTree().inorder();
}

void admin::mostrarDatosEnTabla()
{
    // Mostrar datos en una tabla
    ui->tableBuscar->setRowCount(0);
    int row = 0;
    AppData& appData = AppData::getInstance();

    auto inorderToTable = [&](shared_ptr<Node> node, auto&& inorderToTableRef) -
> void {
        if (!node) return;

        inorderToTableRef(node->left, inorderToTableRef);

        ui->tableBuscar->insertRow(row);
        // Llenar datos en la tabla y agregar botones de modificar y eliminar
        ui->tableBuscar->setItem(row, 0, new
QTableWidgetItem(QString::fromStdString(node->nombre)));
        // Agregar botones con funcionalidades
        QPushButton *btnModificar = new QPushButton("Modificar");
        QPushButton *btnEliminar = new QPushButton("Eliminar");
```

```
        connect(btnModificar, &QPushButton::clicked, [this, node, &appData]() {
            // Lógica para modificar usuario
            this->hide();
            DialogModificar dialog;
            // Establecer datos en el diálogo y actualizar AVL
            int resultado = dialog.exec();
            if (resultado == QDialog::Accepted) {
                // Actualizar usuario en AVL
                appData.getAVLTree().deleteNode(node->email);
                appData.getAVLTree().insert(/* nuevos datos */);
                mostrarDatosEnTabla();
                QMessageBox::information(this, "Modificado", "El usuario ha sido
modificado con éxito.");
            }
        });

        connect(btnEliminar, &QPushButton::clicked, [this, node]() {
            // Lógica para eliminar usuario
            this->eliminarUsuario(node);
        });

        ui->tableBuscar->setCellWidget(row, 4, btnModificar);
        ui->tableBuscar->setCellWidget(row, 5, btnEliminar);

        row++;
        inorderToTableRef(node->right, inorderToTableRef);
    };
```

```
    inorderToTable(appData.getAVLTree().getRoot(), inorderToTable);
}

void admin::eliminarUsuario(shared_ptr<Node> node)
{
    // Eliminar usuario del AVL y actualizar la tabla
    AppData& appData = AppData::getInstance();
    appData.getAVLTree().deleteNode(node->email);
    mostrarDatosEnTabla();
    QMessageBox::information(this, "Eliminado", "El usuario ha sido eliminado.");
}

void admin::on_buscarbtn_clicked()
{
    // Buscar usuario por correo y mostrar en la tabla
    AppData& appData = AppData::getInstance();
    QString correo = ui->searchLine->text();
    std::string email = correo.toStdString();
    ui->tableBuscar->setRowCount(0);
    int row = 0;

    shared_ptr<Node> nodo = appData.getAVLTree().getNode(email);
    if (nodo) {
        ui->tableBuscar->insertRow(row);
        // Llenar datos en la tabla y agregar botones
        QPushButton *btnModificar = new QPushButton("Modificar");
        QPushButton *btnEliminar = new QPushButton("Eliminar");
```

```
        connect(btnModificar, &QPushButton::clicked, [this, nodo]() {
            // Lógica para modificar usuario
            this->hide();
            DialogModificar dialog;
            // Establecer datos en el diálogo y actualizar AVL
            int resultado = dialog.exec();
            if (resultado == QDialog::Accepted) {
                AppData& appData = AppData::getInstance();
                appData.getAVLTree().deleteNode(nodo->email);
                appData.getAVLTree().insert(/* nuevos datos */);
                mostrarDatosEnTabla();
                QMessageBox::information(this, "Modificado", "El usuario ha sido
modificado con éxito.");
            }
        });

        connect(btnEliminar, &QPushButton::clicked, [this, nodo]() {
            // Lógica para eliminar usuario
            this->eliminarUsuario(nodo);
        });

        ui->tableBuscar->setCellWidget(row, 4, btnModificar);
        ui->tableBuscar->setCellWidget(row, 5, btnEliminar);

        QMessageBox::information(this, "Búsqueda exitosa", "El usuario ha sido
encontrado.");
    } else {
        QMessageBox::warning(this, "Error", "Usuario no encontrado en el AVL
global.");
```

```
    }
}

void admin::on_ordenbtn_clicked()
{
    // Mostrar datos en la tabla según el recorrido seleccionado
    QString ordenSeleccionado = ui->ordenBox->currentText();
    ui->tableBuscar->setRowCount(0);
    int row = 0;
    AppData& appData = AppData::getInstance();

    auto addToTable = [&](shared_ptr<Node> node) {
        ui->tableBuscar->insertRow(row);
        ui->tableBuscar->setItem(row, 0, new
QTableWidgetItem(QString::fromStdString(node->nombre)));
        // Agregar botones
        QPushButton *btnModificar = new QPushButton("Modificar");
        QPushButton *btnEliminar = new QPushButton("Eliminar");

        connect(btnModificar, &QPushButton::clicked, [this, node]() {
            // Lógica para modificar usuario
            this->hide();
            DialogModificar dialog;
            // Establecer datos en el diálogo y actualizar AVL
            int resultado = dialog.exec();
            if (resultado == QDialog::Accepted) {
                AppData& appData = AppData::getInstance();
                appData.getAVLTree().deleteNode(node->email);
                appData.getAVLTree().insert(/* nuevos datos */);
```

```
            mostrarDatosEnTabla();
            QMessageBox::information(this, "Modificado", "El usuario ha sido modificado con éxito.");
        }
    });

    connect(btnEliminar, &QPushButton::clicked, [this, node]() {
        // Lógica para eliminar usuario
        this->eliminarUsuario(node);
    });

    ui->tableBuscar->setCellWidget(row, 4, btnModificar);
    ui->tableBuscar->setCellWidget(row, 5, btnEliminar);
    row++;
};

// Ejecutar el recorrido seleccionado
if (ordenSeleccionado == "InOrder") {
    appData.getAVLTree().inorderTraversal([&](shared_ptr<Node> node) { addToTable(node); });
} else if (ordenSeleccionado == "PreOrder") {
    appData.getAVLTree().preorderTraversal([&](shared_ptr<Node> node) { addToTable(node); });
} else if (ordenSeleccionado == "PostOrder") {
    appData.getAVLTree().postorderTraversal([&](shared_ptr<Node> node) { addToTable(node); });
}
}
```

Registro.cpp

Descripción General

El archivo `dialogmodificar.cpp` define la implementación de la clase `DialogModificar`, que es un diálogo para modificar los datos de un usuario. Este diálogo permite al usuario ingresar y actualizar información como nombre, apellido, correo electrónico, fecha de nacimiento y contraseña.

Funcionalidades Clave

1. **Constructor (`DialogModificar::DialogModificar`)**

   - **Propósito**: Inicializa el diálogo y configura la interfaz gráfica.

   - **Implementación**: Llama al constructor base de `QDialog` y luego configura la interfaz de usuario usando `setupUi`.

2. **Destructor (`DialogModificar::~DialogModificar`)**

   - **Propósito**: Libera los recursos asociados con la interfaz de usuario cuando el diálogo se destruye.

   - **Implementación**: Elimina el puntero `ui` que gestiona los elementos gráficos del diálogo.

3. **Setters**: Métodos para establecer valores en los campos de entrada del diálogo.

   - **`setNombre(const QString &nombre)`**: Establece el texto del campo de nombre (`lineEditNombre`) a `nombre`.

   - **`setApellido(const QString &apellido)`**: Establece el texto del campo de apellido (`lineEditApellido`) a `apellido`.

   - **`setEmail(const QString &email)`**: Establece el texto del campo de correo electrónico (`lineEditEmail`) a `email`.

   - **`setFechaNacimiento(const QString &fecha)`**: Establece el texto del campo de fecha de nacimiento (`lineEditFechaNacimiento`) a `fecha`.

- **`setPassword(const QString &password)`**: Establece el texto del campo de contraseña (`lineEditPassword`) a `password`.

4. **Getters**: Métodos para obtener los valores de los campos de entrada del diálogo.

- **`getNombre() const`**: Devuelve el texto del campo de nombre (`lineEditNombre`).

- **`getApellido() const`**: Devuelve el texto del campo de apellido (`lineEditApellido`).

- **`getEmail() const`**: Devuelve el texto del campo de correo electrónico (`lineEditEmail`).

- **`getFechaNacimiento() const`**: Devuelve el texto del campo de fecha de nacimiento (`lineEditFechaNacimiento`).

- **`getPassword() const`**: Devuelve el texto del campo de contraseña (`lineEditPassword`).

CODIGO COMENTADO

```
#include "dialogmodificar.h"      // Incluye la definición de la clase DialogModificar
#include "ui_dialogmodificar.h"  // Incluye la definición de la interfaz de usuario generada

// Constructor de la clase DialogModificar
DialogModificar::DialogModificar(QWidget *parent)
    : QDialog(parent)         // Llama al constructor base de QDialog
    , ui(new Ui::DialogModificar)  // Inicializa el puntero ui con la instancia de la interfaz
{
    ui->setupUi(this);  // Configura la interfaz de usuario del diálogo
}
```

```
// Destructor de la clase DialogModificar
DialogModificar::~DialogModificar()
{
    delete ui;  // Libera la memoria ocupada por el puntero ui
}

// Métodos Setters para establecer valores en los campos del diálogo

// Establece el texto del campo de nombre
void DialogModificar::setNombre(const QString &nombre) {
    ui->lineEditNombre->setText(nombre);  // Asigna el valor a lineEditNombre
}

// Establece el texto del campo de apellido
void DialogModificar::setApellido(const QString &apellido) {
    ui->lineEditApellido->setText(apellido);  // Asigna el valor a lineEditApellido
}

// Establece el texto del campo de correo electrónico
void DialogModificar::setEmail(const QString &email) {
    ui->lineEditEmail->setText(email);  // Asigna el valor a lineEditEmail
}

// Establece el texto del campo de fecha de nacimiento
void DialogModificar::setFechaNacimiento(const QString &fecha) {
    ui->lineEditFechaNacimiento->setText(fecha);  // Asigna el valor a
lineEditFechaNacimiento
}
```

```
// Establece el texto del campo de contraseña
void DialogModificar::setPassword(const QString &password) {
    ui->lineEditPassword->setText(password);  // Asigna el valor a lineEditPassword
}

// Métodos Getters para obtener valores de los campos del diálogo

// Devuelve el texto del campo de nombre
QString DialogModificar::getNombre() const {
    return ui->lineEditNombre->text();  // Retorna el texto del campo lineEditNombre
}

// Devuelve el texto del campo de apellido
QString DialogModificar::getApellido() const {
    return ui->lineEditApellido->text();  // Retorna el texto del campo lineEditApellido
}

// Devuelve el texto del campo de correo electrónico
QString DialogModificar::getEmail() const {
    return ui->lineEditEmail->text();  // Retorna el texto del campo lineEditEmail
}

// Devuelve el texto del campo de fecha de nacimiento
QString DialogModificar::getFechaNacimiento() const {
    return ui->lineEditFechaNacimiento->text();  // Retorna el texto del campo
lineEditFechaNacimiento
}

// Devuelve el texto del campo de contraseña
```

```
QString DialogModificar::getPassword() const {
    return ui->lineEditPassword->text();  // Retorna el texto del campo lineEditPassword
}
```