Manual Técnico

Maria Paola Guadalupe Dávila Valenzuela
Estructura de Datos
Sección A
17/12/2024

REQUISITOS DEL SISTEMA

• Sistema operativo: Windows 10 o superior de 64 bits

• CPU: Intel(R) Core (TM) i5-5300U CPU @ 2.30GHz 2.29 GHz o superior

• Memoria RAM: 8 GB o superior

• Tarjeta gráfica: NVIDIA GeForce GTX 970 (4GB), GTX 1060 (6GB), GTX 1650 (4GB) o AMD Radeon R9 290 (4GB) / RX 470 (4GB)

• Tener abierto Visual Studio Code.

Implementación y Estructura del Código
Este programa ha sido desarrollado principalmente utilizando un lenguaje C++, desarrollándose dentro de la consola, crea un programa en el cual se pueden administrar los activos de cada usuario e interactuar entre ellos.
Clase usuario: Esta clase almacena la información de un usuario, incluyendo su nombre, contraseña y nombre completo.

```
class Usuario {
public:
    string nombre;
    string contrasena;
    string nombre_completo;

    Usuario(string nombre, string contrasena, string nombre_completo) {
        this->nombre = nombre;
        this->contrasena = contrasena;
        this->nombre_completo = nombre_completo;
    }
};
```

Clase nodo: Esta clase representa un nodo en una estructura de datos, que contiene un objeto Usuario y un puntero al siguiente nodo.

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

    Nodo(Usuario usuario) : usuario(usuario), siguiente(nullptr) {}
};

// Definición de la clase Matriz
class Matriz {
public:
    Nodo* cabeza;

    Matriz() : cabeza(nullptr) {}

    void insertar(Nodo* nodo, string empresa, string departamento, Usuario usuario) {
        // Implementación de la inserción en la matriz
        // ...código existente...
    }
};
```

Clase matriz: Esta clase representa una matriz que contiene nodos. Incluye un método para insertar nodos en la matriz.

```
class Matriz {
public:
    Nodo* cabeza;

    Matriz() : cabeza(nullptr) {}

    void insertar(Nodo* nodo, string empresa, string departamento, Usuario usuario) {
        // Implementación de la inserción en la matriz
        // ...código existente...
    }
};
```

Función para insertar usuarios quemados: Esta función inserta una lista de usuarios predefinidos (quemados) en la matriz.

```
void insertarUsuariosQuemados(Matriz* matriz) {
    struct UsuarioData {
        string nombre;
        string nombre_completo;
        string contrasena;
        string departamento;
        string empresa;
    };

    UsuarioData usuarios[] = {
        {"elian_estrada", "Elian Estrada", "1234", "guatemala", "igss"},
        {"juanito", "Juan Perez", "4567", "jutiapa", "max"},
        {"pedrito", "Pedro Rodriguez", "48956", "jalapa", "usac"},
        {"mafer", "Maria Fernanda", "54312", "peten", "cinepolis"},
        {"juanma", "Juan Manuel", "32897", "guatemala", "usac"},
        {"casimiro", "Carlos Perez", "721896", "guatemala", "max"},
        {"fuego03", "Fernando Mendez", "891346", "jutiapa", "cinepolis"},
        {"azurdia", "Alejandra Guzman", "780145", "jutiapa", "usac"},
        {"incrediboy", "Iraldo Martinez", "201598", "jutiapa", "max"},
        {"alcachofa", "Antonio Lopez", "20435", "jalapa", "usac"}
    };

    for (const auto& u : usuarios) {
        Usuario* usuario = new Usuario(u.nombre, u.contrasena, u.nombre_completo);
        matriz->cabeza->insertar(matriz->cabeza, u.empresa, u.departamento, *usuario);
    }
```

Función rentar activo: es la función que se encarga del proceso para la renta de los activos donde se seleccionan mediante el ID.

```
void Menu::rentarActivo() {
    int opcion = 0;
    do {
        cout << "\n=== Rentar activo ===\n";
        matriz->cabeza->mostrarActivosDisponibles(matriz->cabeza);
        cout << "Ingrese una opcion:\n";
        cout << "1. Rentar un activo:\n";
        cout << "2. Regresar al menu:\n";
        opcion = obtenerEntrada();
        if (opcion == 1) {
            cout << "Ingrese el id del activo a rentar:\n";
            string id = obtenerEntradaString();
            Activo* activo = matriz->cabeza->buscarActivo(matriz->cabeza, id);
            if (activo == nullptr) {
                cout << "Id del activo no encontrado\n";
            } else {
                activo->mostrarActivo();
                cout << "Ingrese el tiempo a rentar (ingrese el numero 0 si desea regresar):\n";
                int tiempo = obtenerEntrada();
                if (tiempo > 0) {
                    activo->rentarActivo();
                    Transaccion* transaccion = new Transaccion(id, usuarioActual->getNombre(),
                        departamentoActual, empresaActual, time(nullptr), tiempo);
                    lista->insertar(*transaccion);
                    cout << "Activo rentado\n";
                }
            }
        }
    } while (opcion != 2);
}
```

Función mostrar menu usuario: Presenta el menu principal, ofrece las diferentes opciones y tambien indica que método debe llamar dependiendo de la opción que se seleccione

```
void Menu::mostrarMenuUsuario() {
    int opcion;
    do {
        cout << "\n=== Menu de usuario ===\n";
        cout << "1. Agregar activo\n";
        cout << "2. Eliminar activo\n";
        cout << "3. Modificar activo\n";
        cout << "4. Mostrar activos\n";
        cout << "5. Rentar activo\n";
        cout << "6. Regresar al menu principal\n";
        opcion = obtenerEntrada();
```

Función agregar activo: genera un ID aleatorio para el activo, verifica que el id no se haya registrado antes.

```cpp
void Menu::agregarActivo() {
    cout << "\n=== Agregar activo ===\n";
    string id = generarIdAleatorio(10);
    string descripcion;
    cout << "Ingrese la descripcion del activo:\n";
    descripcion = obtenerEntradaString();
    Activo* activo = new Activo(id, descripcion);
    usuarioActual->activos.insertar(*activo);
    cout << "Activo agregado con ID: " << id << "\n";
}
```

Función eliminar activo: solicita el id del activo que se desea remover, verifica que el archivo exista dentro de la matriz, llama al método para eliminarlo y muestra un mensaje de confirmación.

```cpp
void Menu::eliminarActivo() {
    cout << "\n=== Eliminar activo ===\n";
    cout << "Ingrese el ID del activo a eliminar:\n";
    string id = obtenerEntradaString();
    if (usuarioActual->activos.eliminarPorId(id)) {
        cout << "Activo eliminado\n";
    } else {
        cout << "ID del activo no encontrado\n";
    }
}
```

Función modificar activo: solicita el id del activo que se desea modificar, pide al usuario una nueva descripción para actualizarlo y muestra un mensaje de confirmación.

```cpp
void Menu::modificarActivo() {
    cout << "\n=== Modificar activo ===\n";
    cout << "Ingrese el ID del activo a modificar:\n";
    string id = obtenerEntradaString();
    Activo* activo = usuarioActual->activos.buscarPorId(id);
    if (activo == nullptr) {
        cout << "ID del activo no encontrado\n";
    } else {
        cout << "Ingrese la nueva descripcion del activo:\n";
        string descripcion = obtenerEntradaString();
        activo->setDescripcion(descripcion);
        cout << "Activo modificado\n";
        activo->mostrarActivo();
    }
}
```

Función mostrar activos: Llama a los datos dentro de la matriz, lo recorre y los muestra como los activos disponibles.

```cpp
void Menu::mostrarActivos() {
    cout << "\n=== Mostrar activos ===\n";
    usuarioActual->activos.mostrar();
}
```