Manual Técnico

Fase 3

Laboratorio de Estructuras de Datos
Sección C
Derek Francisco Orellana Ibáñez
202001151

Formato archivo JSON

Usuarios

El archivo usuarios.json debe contener el siguiente formato, nombres, apellidos, fecha_de_nacimiento, correo y contraseña, este debe estar en un arreglo.

```
{
    "nombres": "Carlos",
    "apellidos": "Gomez",
    "fecha_de_nacimiento": "1990/05/12",
    "correo": "user1@example.com",
    "contraseña": "123"
},
```

Solicitudes

El formato del archivo solicitudes.json debe contener un arreglo con objetos con atributos de correo del emisor, receptor y el estado de la solicitud.

```
{
  "emisor": "admin@gmail.com",
  "receptor": "user1@example.com",
  "estado": "ACEPTADA"
},
```

Publicaciones

Debe contener un arreglo de objetos con los siguientes parámetros, correo, contenido, fecha, hora y un arreglo de comentarios los cuales tiene el formato, correo, comentario, fecha y hora.

```
{
    "correo": "admin@gmail.com",
    "contenido": "Disfrutando del paisaje.",
    "fecha": "12/08/2024",
    "hora": "12:45",
    "comentarios": [
        {
            "correo": "user1@example.com",
            "comentario": "¡Qué bonito!",
            "fecha": "13/08/2024",
            "hora": "09:15"
        }
    ]
},
```

Nuevas funcionalidades

Clase Blockchain

Métodos privados

```
ListaEnlazada::ListaEnlazada<Structs::Block> chain;
```

Creacion de una lista enlazada para almacenar los bloques

```
std::string generateTimestamp();
```

Genera un string con la fecha y hora en el momento que crea un bloque

Métodos públicos

```
Blockchain();
```

Constructor de la clase Bockchain la cual se encarga de almacenar los bloques en una lista

```
void addBlock(Structs::Block newBlock);
```

metodo para agregar un nuevo bloque a la cadena, agregándole las propiedades faltantes del bloque como: nonce, rootHash, hash, prevHash y validate

```
void exportBlocks();
```

Exporta los bloques en archivos json dentro de la carpeta **Blockchain/**

```
void importBlocks();
```

Importa los bloques en la carpeta **Blockchain/** y los valida

```
void validateBlock(int index);
```

Valida un bloque de la lista

```
void validateBlocks();
```

Valida todos los bloques de la cadena

```
string graficar();
```

Grafica los bloques generados en la cadena

```
ListaEnlazada::ListaEnlazada<Structs::Block> getChain();
```

Retorna la lista de bloques actuales en la cadena

Funciones nuevas

```
std::string leerArchivo(const std::string& path);
```

Metodo que se encarga de leer los archivos .edd y .json para retornar el string

```
void cargarEstructuras(const std::string& data);
```

metodo que se encarga de cargar las estructuras pasando como parámetro la data como string

```
void agregarSeguridad();
```

Metodo que se encarga de generar un nuevo bloque en la cadena, agregada cuando se crea, modifica o se comenta una publicación

```
int obtenerCommentID(int postid);
```

metodo para obtener el id de un comentario

```
bool existeComentario(int postid, int cid);
```

metodo para verificar si existe un comentario

```
std::string generateTimestamp();
```

metodo que se encarga de generar un timestamp

Globales

```
extern Blockchain::Blockchain seguridad_blockchain;
```

Se agrego una clase global para almacenar los bloques del blockchain

Nuevos Structs

```
struct AmigoSugerido
```

Struct que contiene un Struct Usuario y cantidad de amigos en comun

```
struct Block
```

Struct que contiene un índice, el timestamp, una lista de publicaciones data, el rootHash, el prevHash, nonce, validate y el hash del bloque.

```
void calculateNonceAndHash()
```

Método encargado de generar el nonce y el hash del bloque cumpliendo con la condición de que debe empezar por cuatro ceros

```
std::string getJsonData()
```

Método que se encarga de retornar el string del json generado de todos sus datos

```
void validateBlock(bool val)
```

Método que cambia la validación del bloque

GrafoRelacion

Métodos privados

```
void insertarNodo(Structs::Usuario &usuario);
```

Método para insertar un nuevo nodo si no existe

```
Nodo *buscarNodo(const std::string &correo) const;
```

Buscar un nodo por el correo del usuario

Métodos públicos

```
GrafoRelacion() {}
```

Constructor de la clase

```
void agregarRelacion(Structs::Usuario &user1, Structs::Usuario &user2);
```

Metodo que agrega la relación entre dos usuarios

```
void eliminarRelacion(const std::string &correo1, const std::string &correo2);
```

Metodo que elimina la relación entre 2 usuarios

```
void eliminarRelacionesUsuario(const std::string &correo);
```

Método que elimina todas las relaciones del usuario

```
ListaEnlazada::ListaEnlazada<Structs::Usuario> obtenerAmigos(const std::string &correo);
```

Método que obtiene la lista de amigos del usuario

```
ListaEnlazada::ListaEnlazada<Structs::AmigoSugerido> sugerirAmigos(const std::string
&correo);
```

Método que obtiene la lista de amigos sugeridos del usuario

```
bool verificarRelacion(const std::string &correo1, const std::string &correo2);
```

Metodo que verifica la relación de dos usuarios

```
bool estaVacio() const;
```

método que verifica si hay amistades en el grafo

```
std::string graficarGrafo();
```

método que grafica el grafo en general

```
std::string graficarGrafo(const std::string &correo);
```

método sobrecargado que grafica el grafo de un usuario

```
std::string graficarGrafoSugeridos(const std::string &correo);
```

método que grafica el grafo de amistades sugeridas de un usuario

```
std::string tablaAdyacencia();
```

método que grafica la tabla de adyacencia de un usuario

Huffman

Métodos privados

```
void insert(PriorityQueue*& head, Node* newNode);
```

Metodo que inserta un nodo de prioridad

```
Node* extractMin(PriorityQueue*& head);
```

Metodo que extrae la prioridad mínima de la lista

```
void buildHuffmanCode(Node* root, std::string code, std::string codes[256]);
```

Metodo que construye los códigos de huffman

```
std::string compress(const std::string& text, std::string codes[256]);
```

Metodo que comprime la información

```
void generateDotFile(Node* root, std::ofstream& dotFile, int& nullCount);
```

Metodo que genera el archivo dotfile para graficar

```
Node* root;
```

Noto raiz

```
void serialize(Node* node, nlohmann::json& jsonNode);
```

Metodo que serialize la informacion

```
Node* deserialize(const nlohmann::json& jsonNode);
```

Metodo que deserializa la información

Métodos públicos

```
std::string compress(const std::string& text);
```

Metodo que comprime la información data

```
std::string decompress(const std::string& compressedText);
```

Metodo que descomprime la información data

```
void createGraph(Node* root);
```

Metodo que grafica el árbol de huffman pasandole la raiz

```
Node* getRoot();
```

Método que obtiene la raíz

```
void exportTree(const std::string& filename);
```

Metodo que exporta el árbol de huffman

```
bool importTree(const std::string& filename);
```

Metodo que importa el árbol de huffman

Merkle

Metodos privados

```
Nodo *root;
```

Nodo raiz del árbol de merkle

```
Nodo* combineNodes(Nodo* left, Nodo *right);
```

Combinación de nodos para generar el árbol de merkle

```
Nodo* buildMerkleTree(Nodo** leaves,int start, int end);
```

Metodo que se encarga de construir el árbol de merkle

```
void generateDotFile(Nodo* node, ofstream &outFile,int& nodeCounter) const ;
```

Metodo que genera los nodos del archivo dotfile

```
void generateDatablockConnections(Nodo* node, ofstream &outFile, int& nodeCounter);
```

Metodo que genera las conexiones de los datablos con los nodos

```
ListaEnlazada::ListaEnlazada<Structs::Publicacion> data_temp;
```

Método que retorna la lista temporal del bloque

Métodos públicos

```
Merkle();
```

Constructor general

```
Merkle(ListaEnlazada::ListaEnlazada<Structs::Publicacion> &data);
```

Constructor que solicita la lista de publicaciones del bloque

```
~Merkle();
```

Destructor

```
string getRootHash() const;
```

Metodo que retorna el hash de la raíz

```
std::string graficar();
```

Método que grafica el árbol de merkle

Clases anteriores

Clase ArbolABB

Constructor por defecto del árbol binario de búsqueda.

```
ArbolABB();
```

Destructor que limpia el árbol.

```
~ArbolABB();
```

Métodos públicos

Inserta una publicación en el nodo correspondiente a la fecha.

```
void insertar(const std::tm &fecha, const Structs::Publicacion &publicacion);
```

Busca un nodo con la fecha dada.

```
Nodo *buscar(const std::tm &fecha) const;
```

Genera una representación visual del árbol en formato Graphviz.

```
string graficar();
```

Limpia el árbol, eliminando todos los nodos.

```
void limpiar();
```

Retorna una lista con las fechas almacenadas en el árbol.

```
ListaEnlazada::ListaEnlazada<std::tm> obtenerFechas() const;
```

Retorna una lista de publicaciones asociadas a una fecha específica.

```
ListaEnlazada::ListaEnlazada<Structs::Publicacion> obtenerPublicaciones(const std::tm
&fecha) const;
```

Retorna publicaciones asociadas a una fecha, orden y cantidad.

```
ListaEnlazada::ListaEnlazada<Structs::Publicacion> obtenerPublicaciones(const std::tm
&fecha, const int orden, int cantidad);
```

Retorna una lista de publicaciones en recorrido inorder con un límite de cantidad.

```
ListaEnlazada::ListaEnlazada<Structs::Publicacion> inorder(const int cantidad);
```

Retorna una lista de publicaciones en recorrido preorder con un límite de cantidad.

```
ListaEnlazada::ListaEnlazada<Structs::Publicacion> preorder(const int cantidad);
```

Retorna una lista de publicaciones en recorrido postorder con un límite de cantidad.

```
ListaEnlazada::ListaEnlazada<Structs::Publicacion> postorder(const int cantidad);
```

Retorna una lista de nodos ordenados por la cantidad de publicaciones.

```
ListaEnlazada::ListaEnlazada<Nodo *> obtenerNodosOrdenados() const;
```

Métodos privados

Destruye el árbol recursivamente.

```
void destruirArbol(Nodo *nodo);
```

Inserta un nodo en el árbol.

```
Nodo *insertar(Nodo *nodo, const std::tm &fecha, const Structs::Publicacion &publicacion);
```

Busca un nodo por su fecha.

```
Nodo *buscar(Nodo *nodo, const std::tm &fecha) const;
```

Genera el archivo .dot para visualización en Graphviz.

```
void generarDot(Nodo *nodo, std::ostream &out) const;
```

Obtiene publicaciones de un nodo según la fecha.

```
ListaEnlazada::ListaEnlazada<Structs::Publicacion> obtenerPublicaciones(Nodo *nodo, const std::tm &fecha) const;
```

Obtiene publicaciones según fecha, orden y cantidad.

```
ListaEnlazada::ListaEnlazada<Structs::Publicacion> obtenerPublicaciones(Nodo *nodo, const std::tm &fecha, const int orden, int cantidad) const;
```

Realiza recorrido inorder para obtener una lista de publicaciones.

```
void inOrdenRecursivo(Nodo *nodo, ListaEnlazada::ListaEnlazada<Structs::Publicacion> &lista);
```

Realiza recorrido inorder para una fecha específica.

```
void inOrdenRecursivo(Nodo *nodo, const std::tm &fecha, ListaEnlazada::ListaEnlazada<Structs::Publicacion> &lista);
```

Realiza recorrido preorder para obtener una lista de publicaciones.

```
void preOrdenRecursivo(Nodo *nodo, ListaEnlazada::ListaEnlazada<Structs::Publicacion> &lista);
```

Realiza recorrido preorder para una fecha específica.

```
void preOrdenRecursivo(Nodo *nodo, const std::tm &fecha, ListaEnlazada::ListaEnlazada<Structs::Publicacion> &lista);
```

Realiza recorrido postorder para obtener una lista de publicaciones.

```
void postOrdenRecursivo(Nodo *nodo, ListaEnlazada::ListaEnlazada<Structs::Publicacion> &lista);
```

Realiza recorrido postorder para una fecha específica.

```
void postOrdenRecursivo(Nodo *nodo, const std::tm &fecha,
ListaEnlazada::ListaEnlazada<Structs::Publicacion> &lista);
```

Obtiene las fechas de los nodos en el árbol.

```
ListaEnlazada::ListaEnlazada<std::tm> obtenerFechas(Nodo *nodo) const;
```

Clase ArbolAVL

Constructor por defecto del árbol AVL.

```
ArbolAVL();
```

Destructor que limpia el árbol.

```
~ArbolAVL();
```

Métodos públicos

Inserta un usuario en el árbol AVL.

```
bool insertar(const Structs::Usuario &usuario);
```

Elimina un usuario del árbol AVL según su correo.

```
bool eliminar(const string &correo);
```

Modifica los datos de un usuario, excluyendo el rol.

```
bool modificar(const std::string &correo, std::string nombres, std::string apellidos, std::string
fecha, std::string contrasena);
```

Modifica los datos de un usuario, incluyendo el rol.

```
bool modificar(const std::string &correo, std::string nombres, std::string apellidos, std::string
fecha, std::string contrasena, std::string rol);
```

Busca un usuario por su correo electrónico.

```
Structs::Usuario *buscar(const string &correo);
```

Obtiene el ID del usuario.

```
int obtenerId();
```

Retorna una lista de usuarios en recorrido inorder.

```
ListaEnlazada::ListaEnlazada<Structs::Usuario> InOrder();
```

Retorna una lista de usuarios en recorrido preorder.

```
ListaEnlazada::ListaEnlazada<Structs::Usuario> PreOrder();
```

Retorna una lista de usuarios en recorrido postorder.

```
ListaEnlazada::ListaEnlazada<Structs::Usuario> PostOrder();
```

Genera una representación gráfica del árbol en formato Graphviz.

```
string graficar();
```

Elimina las solicitudes del usuario logueado según su correo.

```
bool eliminarSolicitudes(const std::string &correo);
```

Envía una solicitud de amistad de un usuario a otro.

```
bool enviarSolicitud(const std::string &correoEmisor, const std::string &correoReceptor);
```

Cancela una solicitud de amistad entre dos usuarios.

```
bool cancelarSolicitud(const std::string &correoEmisor, const std::string &correoReceptor);
```

Rechaza una solicitud de amistad recibida.

```
bool rechazarSolicitud(const std::string &correoEmisor, const std::string &correoReceptor);
```

Acepta una solicitud de amistad.

```
bool aceptarSolicitud(const std::string &correoEmisor, const std::string &correoReceptor);
```

Métodos privados

Destruye el árbol recursivamente.

```
void destruirArbolRecursivo(Nodo *nodo);
```

Obtiene la altura de un nodo.

```
int obtenerAltura(Nodo *nodo);
```

Calcula el balance de un nodo.

```
int obtenerBalance(Nodo *nodo);
```

Inserta un nodo en el árbol AVL.

```
Nodo *insertarNodo(Nodo *nodo, const Structs::Usuario &usuario);
```

Elimina un nodo del árbol AVL según el correo.

```
Nodo *eliminarNodo(Nodo *nodo, const string &correo);
```

Busca un nodo en el árbol por correo.

```
Nodo *buscarNodo(Nodo *nodo, const string &correo);
```

Realiza una rotación a la izquierda para balancear el árbol.

```
Nodo *rotacionIzquierda(Nodo *nodo);
```

Realiza una rotación a la derecha para balancear el árbol.

```
Nodo *rotacionDerecha(Nodo *nodo);
```

Balancea un nodo del árbol.

```
Nodo *balancearNodo(Nodo *nodo);
```

Busca el nodo con el valor mínimo en el subárbol.

```
Nodo *nodoConValorMinimo(Nodo *nodo);
```

Realiza recorrido inorder recursivo y llena la lista con usuarios.

```
void inOrdenRecursivo(Nodo *nodo, ListaEnlazada::ListaEnlazada<Structs::Usuario> &lista);
```

Realiza recorrido preorder recursivo y llena la lista con usuarios.

```
void preOrdenRecursivo(Nodo *nodo, ListaEnlazada::ListaEnlazada<Structs::Usuario> &lista);
```

Realiza recorrido postorder recursivo y llena la lista con usuarios.

```
void postOrdenRecursivo(Nodo *nodo, ListaEnlazada::ListaEnlazada<Structs::Usuario> &lista);
```

Genera un archivo .dot para visualización en Graphviz del nodo.

```
void graficarNodo(Nodo *nodo, std::ofstream &archivoDot);
```

Obtiene el usuario con el mayor ID del árbol.

```
Structs::Usuario obtenerUsuarioConMayorID();
```

Clase ArbolBST

Constructor por defecto del árbol binario de búsqueda (BST).

```
ArbolBST();
```

Destructor que limpia los nodos del árbol.

```
~ArbolBST();
```

Métodos públicos

Agrega un nuevo valor ReportePosts al árbol.

```
void add(Structs::ReportePosts val);
```

Realiza un recorrido preorder del árbol.

```
void preorder();
```

Realiza un recorrido inorder del árbol.

```
void inorder();
```

Realiza un recorrido postorder del árbol.

```
void postorder();
```

Busca un reporte por fecha en el árbol.

```
void buscarPorFecha(string fecha);
```

Genera una representación gráfica del árbol en formato Graphviz.

```
string graficar(string fecha);
```

Métodos privados

Agrega un nuevo valor ReportePosts a un nodo específico del árbol.

```
void add(Structs::ReportePosts val, Node* tmp);
```

Realiza un recorrido preorder de manera recursiva a partir de un nodo.

```
void preorder(Node* tmp);
```

Realiza un recorrido inorder de manera recursiva a partir de un nodo.

```
void inorder(Node* tmp);
```

Realiza un recorrido postorder de manera recursiva a partir de un nodo.

```
void postorder(Node* tmp);
```

Realiza un recorrido por niveles del árbol a partir de un nodo.

```
void levelorder(Node* tmp);
```

Busca un reporte por fecha en un nodo específico del árbol.

```
void buscarPorFecha(string fecha, Node* tmp);
```

Genera un archivo .dot para visualización en Graphviz a partir de un nodo y una fecha específica.

```
void graficar(Node* tmp, string fecha, string &dot);
```

Clase DialogModificar

Constructor que recibe un correo y un widget padre opcional.

```
DialogModificar(const std::string correo, QWidget *parent = nullptr);
```

Destructor que libera los recursos del diálogo.

```
~DialogModificar();
```

Métodos públicos

Establece el correo en el diálogo.

```
void setCorreo(const std::string correo);
```

Métodos privados

Llena los datos del diálogo según el correo proporcionado.

```
void llenarDatos(const std::string correo);
```

Slots

Slot activado cuando se acepta el diálogo (botón OK).

```
void on_buttonBox_accepted();
```

Clase DialogNuevoPost

Constructor por defecto que recibe un widget padre opcional.

```
DialogNuevoPost(QWidget *parent = nullptr);
```

Constructor que recibe un ID y un widget padre opcional.

```
DialogNuevoPost(const int id, QWidget *parent = nullptr);
```

Destructor que libera los recursos del diálogo.

```
~DialogNuevoPost();
```

Métodos públicos

Establece el ID del post en el diálogo.

```
void setID(const int id);
```

Obtiene el ID del post.

```
int getID() const;
```

Obtiene el path de la imagen asociada al post.

```
std::string getPathImg();
```

Establece el path de la imagen asociada al post.

```
void setPathImg(const std::string path);
```

Métodos privados

Llena los datos del diálogo basándose en el ID proporcionado.

```
void llenarDatos(const int id);
```

Slots

Slot activado cuando se acepta el diálogo (botón OK).

```
void on_buttonBox_accepted();
```

Slot activado al hacer clic en el botón de imagen.

```
void on_imagenButton_clicked();
```

Clase DialogPost

Constructor que recibe un ID y un widget padre opcional.

```
DialogPost(const int id, QWidget *parent = nullptr);
```

Destructor que libera los recursos del diálogo.

```
~DialogPost();
```

Métodos públicos

Establece el ID del post en el diálogo.

```
void setID(const int id);
```

Obtiene el ID del post.

```
int getID() const;
```

Actualiza los comentarios en el diálogo.

```
void actualizarComentarios();
```

Métodos privados

Llena los datos del diálogo basándose en el ID proporcionado.

```
void llenarDatos(const int id);
```

Slots

Slot activado al hacer clic en el botón para mostrar el árbol de comentarios.

```
void on_pushButton_arbol_comentarios_clicked();
```

Slot activado al hacer clic en el botón para comentar.

```
void on_pushButton_comentar_clicked();
```

Slot activado al hacer clic en el botón para editar.

```
void on_pushButton_editar_clicked();
```

Slot activado al hacer clic en el botón para eliminar.

```
void on_pushButton_eliminar_clicked();
```

Namespace Func

Variables globales

Tabla para gestionar usuarios en la vista de administrador.

```
extern QTableWidget* adminTablaUsuarios;
```

Tablas para solicitudes en la vista de usuario.

```
extern QTableWidget* userTablaUsuarios;

extern QTableWidget* userTablaEnviadas;

extern QTableWidget* userTablaRecibidas;
```

Componentes para gestionar publicaciones del usuario.

```
extern QScrollArea* userPostFeed;

extern QScrollArea* userFriends;

extern QComboBox* selectedDate;

extern QComboBox* selectedOrder;

extern QSpinBox* countPost;
```

Métodos públicos

Inicia sesión con el correo y contraseña proporcionados.

```
void IniciarSesion(string email, string password);
```

Cierra sesión del usuario actual.

```
void CerrarSesion();
```

Actualiza la tabla de usuarios en la vista de administrador.

```
void ActualizarTablaUsuariosAdmin(QTableWidget* table,
ListaEnlazada::ListaEnlazada<Structs::Usuario> &lista);
```

Carga usuarios desde un directorio especificado.

```
void CargarUsuarios(string directorio);
```

Carga solicitudes desde un directorio especificado.

```
void CargarSolicitudes(string directorio);
```

Carga publicaciones desde un directorio especificado.

```
void CargarPublicaciones(string directorio);
```

Elimina la cuenta del usuario con el correo especificado.

```
void EliminarCuenta(string correo);
```

Actualiza la tabla de administrador según la opción seleccionada.

```
void ActualizarTablaAdmin(int opcion);
```

Genera un gráfico de las publicaciones.

```
string graficarPublicaciones();
```

Obtiene el ID de un post.

```
int obtenerPostID();
```

Elimina una publicación con el ID especificado.

```
void eliminarPublicacion(int id);
```

Busca una publicación por ID y devuelve un puntero a ella.

```
Structs::Publicacion *buscarPost(int id);
```

Modifica una publicación con el ID y contenido proporcionados.

```
void modificarPublicacion(int id, std::string contenido, std::string pathImg);
```

Comenta en una publicación con el ID especificado.

```
void ComentarPublicacion(int id, StructsComment::Comentario comentario);
```

Actualiza la lista de fechas.

```
void actualizarListaFechas();
```

Actualiza el árbol de publicaciones.

```
void actualizarArbolPost();
```

Actualiza el feed de publicaciones.

```
void ActualizarFeed();
```

Actualiza la tabla de usuarios.

```
void ActualizarTablaUsuarios(QTableWidget* table);
```

Actualiza la tabla de solicitudes recibidas.

```
void ActualizarTablaRecibidos(QTableWidget* table);
```

Actualiza la tabla de solicitudes enviadas.

```
void ActualizarTablaEnviados(QTableWidget* table);
```

Actualiza todas las tablas de usuario.

```
void ActualizarTablas();
```

Obtiene un reporte de fechas de publicaciones.

```
ListaEnlazada::ListaEnlazada<Structs::ReportePosts> obtenerReporteFechasPost();
```

Obtiene un reporte de fechas de publicaciones de amigos.

```
ListaEnlazada::ListaEnlazada<Structs::ReportePosts> obtenerReporteFechasPostFriends();
```

Obtiene las publicaciones del usuario actual.

```
ListaEnlazada::ListaEnlazada<Structs::Publicacion> obtenerMisPosts();
```

Ordena una lista de reportes de publicaciones utilizando Bubble Sort.

```
void bubbleSort(ListaEnlazada::ListaEnlazada<Structs::ReportePosts> &lista);
```

Ordena una lista de publicaciones utilizando Bubble Sort.

```
void bubbleSort(ListaEnlazada::ListaEnlazada<Structs::Publicacion> &lista);
```

Elimina la cuenta del usuario logeado.

```
void EliminarMiCuenta();
```

Actualiza la lista de amigos del usuario.

```
void ActualizarListaAmigos();
```

Obtiene una lista de usuarios logueados.

```
ListaEnlazada::ListaEnlazada<Structs::Usuario> obtenerListaUsuariosLogeado();
```

Elimina una publicación de un usuario especificado.

```
void eliminarPublicacionUsuario(std::string correo);
```

Convierte una fecha en formato string a otro formato.

```
string convertirFecha(const std::string& fechaOriginal);
```

Convierte una fecha en formato string a un objeto tm.

```
tm convertirFechaTm(const std::string& fechaOriginal);
```

Convierte una fecha en formato string a una cadena de fecha y hora.

```
string convertirFechayHora(const std::string& fechaOriginal);
```

Convierte una fecha en formato tm a una cadena de fecha y hora.

```
string convertirFechayHora(const std::tm &fecha);
```

Convierte una fecha y hora en formato string a un objeto tm.

```
tm convertirFechayHoraTm(const std::string& fechaOriginal);
```

Variables globales

Puntero al usuario actualmente logueado en la aplicación.

```
extern Structs::Usuario *usuario_logeado;
```

Árbol AVL que contiene la lista de usuarios registrados.

```
extern ArbolAVL lista_usuarios;
```

Lista doblemente enlazada que almacena las publicaciones.

```
extern ListaDoble::ListaDoble<Structs::Publicacion> lista_publicaciones;
```

Matriz de relaciones que gestiona las amistades entre usuarios.

```
extern MatrizRelacion relaciones_amistad;
```

Namespace ListaDoble

Métodos privados

Clona los nodos de otra lista doble.

```
void clonar(const ListaDoble &otra)
```

Constructor y Destructor

Constructor que inicializa la lista como vacía.

```
ListaDoble();
```

Destructor que libera la memoria ocupada por los nodos.

```
~ListaDoble();
```

Constructor de copia que clona otra lista doble.

```
ListaDoble(const ListaDoble &otra);
```

Operador de asignación que asigna otra lista doble a la actual.

```
ListaDoble &operator=(const ListaDoble &otra);
```

Métodos públicos

Inserta un valor al final de la lista.

```
void insertar(const T &valor);
```

Inserta un valor al inicio de la lista.

```
void insertarInicio(const T &valor);
```

Elimina el primer nodo de la lista.

```
void eliminarInicio();
```

Elimina el último nodo de la lista.

```
void eliminarFinal();
```

Elimina el nodo en una posición específica.

```
void eliminarPosicion(int pos);
```

Retorna el tamaño actual de la lista.

```
int size() const;
```

Obtiene el nodo según su posición.

```
T *obtener(int pos) const;
```

Verifica si la lista está vacía.

```
bool vacia() const;
```

Namespace ListaEnlazada

Métodos privados

Clona los nodos de otra lista enlazada.

```
Nodo *clonar(Nodo *otraCabeza)
```

Constructor y Destructor

Constructor que inicializa la lista como vacía.

```
ListaEnlazada();
```

Destructor que libera la memoria ocupada por los nodos.

```
~ListaEnlazada();
```

Constructor de copia que clona otra lista enlazada.

```
ListaEnlazada(const ListaEnlazada &otra);
```

Operador de asignación que asigna otra lista enlazada a la actual.

```
ListaEnlazada &operator=(const ListaEnlazada &otra);
```

Métodos públicos

Inserta un nuevo nodo al final de la lista.

```
void insertar(const T &data);
```

Inserta un nuevo nodo al inicio de la lista.

```
void insertarInicio(const T &data);
```

Elimina el primer nodo de la lista.

```
void eliminarInicio();
```

Elimina el último nodo de la lista.

```
void eliminarFinal();
```

Elimina un nodo según su posición.

```
void eliminar(int pos);
```

Retorna el tamaño actual de la lista.

```
int size() const;
```

Obtiene el nodo según su posición.

```
T *obtener(int pos);
```

Verifica si la lista está vacía.

```
bool estaVacia() const;
```

Concatena dos listas enlazadas.

```
void concatenar(const ListaEnlazada &otra);
```

Intercambia dos nodos en posiciones especificadas.

```
void intercambiar(int pos1, int pos2);
```

Métodos de pila

Inserta un nuevo nodo al inicio de la lista (PUSH).

```
void push(const T &data);
```

Elimina el primer nodo de la lista (POP).

```
void pop();
```

Obtiene el valor del primer nodo (TOPE de la pila).

```
T top() const;
```

Clase MatrizRelacion

Constructor Inicializa la matriz de relaciones.

MatrizRelacion():

Destructor Libera los recursos utilizados por la matriz.

~MatrizRelacion():

Métodos públicos

Agrega una relación entre dos usuarios.

void agregarRelacion(Structs::Usuario *user1, Structs::Usuario *user2):

Eliminar relaciones de un usuario

void eliminarRelacionesUsuario(string correo_usuario):

Elimina la relación entre dos usuarios.

void eliminarRelacionEntreUsuarios(string correo1, string correo2):

Verifica si existe una relación entre dos usuarios.

bool verificarRelacion(string correo1, string correo2):

Genera una representación gráfica de las relaciones.

std::string graficar():

Devuelve una lista de amigos de un usuario específico.

ListaEnlazada::ListaEnlazada<Structs::Usuario> obtenerAmigos(string correo):

Verificar si la matriz está vacía

bool estaVacio():

Estructuras | Usuario

Atributos

Identificador único del usuario.

int id

Nombres del usuario.

string nombres

Apellidos del usuario.

string apellidos

Fecha de nacimiento del usuario.

string fechaNacimiento

Correo electrónico del usuario.

string correo

Contraseña del usuario.

string contrasena

Rol del usuario (por defecto, "user").

string rol

Lista de solicitudes de amistad enviadas.

ListaEnlazada::ListaEnlazada<Usuario> solicitudesEnviadas

Lista de solicitudes de amistad recibidas.

ListaEnlazada::ListaEnlazada<Usuario> solicitudesRecibidas

Métodos

Constructor.

Usuario(int id, string nombres, string apellidos, string fechaNacimiento, string correo, string contrasena, string rol = "user")

Muestra la información del perfil del usuario.

void mostrarPerfil() const

Modifica los atributos del usuario.

void modificar(...)

Envía una solicitud de amistad a otro usuario.

void enviarSolicitud(const Usuario &u)

Cancela una solicitud de amistad enviada.

```
void cancelarSolicitud(const string correo)
```

Recibe una solicitud de amistad.

```
void recibirSolicitud(const Usuario &u)
```

Acepta una solicitud de amistad recibida.

```
void aceptarSolicitudRec(const string correo)
```

Acepta una solicitud de amistad enviada.

```
void aceptarSolicitudEnv(const string correo)
```

Rechaza una solicitud de amistad recibida.

```
void rechazarSolicitud(const string correo)
```

Verifica si hay una solicitud de amistad recibida.

```
bool verificarSolicitudRecibida(const string correo)
```

Verifica si hay una solicitud de amistad enviada.

```
bool verificarSolicitudEnviada(const string correo)
```

Estructuras | Publicacion

Atributos

Identificador único de la publicación.

int id

Correo del autor de la publicación.

string correo_autor

Contenido de la publicación.

string contenido

Fecha de creación de la publicación.

string fecha

Hora de creación de la publicación.

string hora

Ruta de la imagen asociada (por defecto, "default.jpg").

string imagen

Árbol B que almacena los comentarios de la publicación.

ArbolB5 *comentarios

Métodos

Constructor.

Publicacion(int id, string correo_autor, const string &contenido, const string &fecha, const string &hora, const string imagen = "default.jpg")

Destructor.

~Publicacion()

Estructuras | ReportePosts

Atributos

Fecha del reporte.

string fecha

Lista de publicaciones en el reporte.

ListaEnlazada::ListaEnlazada<Structs::Publicacion> publicaciones

Métodos

Constructor.

```
ReportePosts(string fecha, ListaEnlazada::ListaEnlazada<Structs::Publicacion> publicaciones)
```

Destructor.

```
~ReportePosts()
```

Estructura | Comentario

Atributos

Fecha y hora en que se realizó el comentario.

```
std::string fecha_hora:
```

Usuario que hizo el comentario.

```
std::string usuario:
```

Contenido del comentario.

```
std::string texto:
```

Métodos

Constructor por defecto que inicializa los atributos a cadenas vacías.

```
Comentario();
```

Constructor que inicializa los atributos con los valores proporcionados.

```
Comentario(const std::string& fecha_hora, const std::string& usuario, const std::string& texto):
```

Destructor por defecto

```
~Comentario():
```

Clase: WidgetComment

Constructor de la clase WidgetComment. Inicializa el widget con un comentario dado.

```
*WidgetComment(const StructsComment::Comentario &c, QWidget parent = nullptr)
```

Destructor de la clase WidgetComment. Libera recursos.

```
~WidgetComment()
```

Métodos Públicos

Establece el comentario en el widget.

```
void setComment(const StructsComment::Comentario &c)
```

Obtiene el comentario del widget.

```
StructsComment::Comentario getComment() const
```

Métodos Privados

Llena los datos del widget con la información del comentario proporcionado.

```
void llenarDatos(const StructsComment::Comentario &c)
```

Clase: WidgetFriend

Constructor de la clase WidgetFriend. Inicializa el widget con un usuario dado.

```
*WidgetFriend(const Structs::Usuario &user, QWidget parent = nullptr)
```

Destructor de la clase WidgetFriend. Libera recursos.

```
~WidgetFriend()
```

Métodos Públicos

Establece el usuario en el widget.

```
void setUser(const Structs::Usuario &u)
```

Obtiene el usuario del widget.

```
Structs::Usuario getUser() const
```

Métodos Privados

Llena los datos del widget con la información del usuario proporcionado.

```
void llenarDatos(const Structs::Usuario &user)
```

Maneja el evento de clic en el botón de eliminar.

```
void on_pushButton_eliminar_clicked()
```

Clase: WidgetPost

Constructor de la clase WidgetPost. Inicializa el widget con un ID de publicación dado.

```
*WidgetPost(const int id, QWidget parent = nullptr)
```

Destructor de la clase WidgetPost. Libera recursos.

```
~WidgetPost()
```

Métodos Públicos

Establece el ID de la publicación en el widget.

```
void setID(const int id)
```

Obtiene el ID de la publicación del widget.

```
int getID() const
```

Métodos Privados

Llena los datos del widget con la información de la publicación proporcionada.

```
void llenarDatos(const int id)
```

Maneja el evento de clic en el botón de ver.

```
void on_pushButton_ver_clicked()
```

Maneja el evento de clic en el botón de eliminar.

```
void on_eliminarButton_clicked()
```

Maneja el evento de clic en el botón de editar.

```
void on_editarButton_clicked()
```

Clase: AdminWindow

Constructor de la clase AdminWindow. Inicializa la ventana principal del administrador.

```
*AdminWindow(QWidget parent = nullptr)
```

Destructor de la clase AdminWindow. Libera recursos.

```
~AdminWindow()
```

Métodos Privados

Maneja el evento de cierre de sesión desde el menú.

```
void on_actionCerrar_Sesion_triggered()
```

Maneja el evento para cambiar a la vista de usuario desde el menú.

```
void on_actionVista_Usuario_triggered()
```

Maneja el evento de clic en el botón de buscar.

```
void on_pushButton_buscar_buscar_clicked()
```

Maneja el evento de clic en el botón de aplicar búsqueda.

```
void on_pushButton_buscar_aplicar_clicked()
```

Maneja el evento de retorno presionado en el campo de búsqueda.

```
void on_lineEdit_buscar_buscar_returnPressed()
```

Maneja el evento de clic en el botón de cargar usuarios.

```
void on_pushButton_cargar_usuarios_clicked()
```

Maneja el evento de clic en el botón de cargar solicitudes.

```
void on_pushButton_cargar_solicitudes_clicked()
```

Maneja el evento de clic en el botón de cargar publicaciones.

```
void on_pushButton_cargar_publicaciones_clicked()
```

Maneja el evento de clic en el botón de generar reportes.

```
void on_pushButton_reportes_generar_clicked()
```

Maneja el evento de clic en el botón de reporte de usuarios.

```
void on_pushButton_reporte_users_clicked()
```

Maneja el evento de clic en el botón de reporte de publicaciones.

```
void on_pushButton_reporte_posts_clicked()
```

Maneja el evento de clic en el botón de reporte de amistades.

```
void on_pushButton_reporte_amistades_clicked()
```

Clase: MainWindow

Constructor de la clase MainWindow. Inicializa la ventana principal de la aplicación.

```
*MainWindow(QWidget parent = nullptr)
```

Destructor de la clase MainWindow. Libera recursos.

```
~MainWindow()
```

Métodos Privados

Maneja el evento de clic en el botón de inicio de sesión.

```
void on_loginButton_clicked()
```

Maneja el evento de retorno presionado en el campo de contraseña.

```
void on_passwordLineEdit_returnPressed()
```

Maneja el evento de retorno presionado en el campo de correo electrónico.

```
void on_emailLineEdit_returnPressed()
```

Maneja el evento de clic en el botón de registro.

```
void on_RegisterButton_clicked()
```

Clase: WindowRegister

Constructor de la clase WindowRegister. Inicializa la ventana de registro de la aplicación.

```
*WindowRegister(QWidget parent = nullptr)
```

Destructor de la clase WindowRegister. Libera recursos.

```
~WindowRegister()
```

Métodos Privados

Maneja el evento de clic en el botón de registrar.

```
void on_registrarButton_clicked()
```

Maneja el evento de retorno presionado en el campo de confirmación de contraseña.

```
void on_confirmarContrasenaLineEdit_returnPressed()
```

Maneja el evento de retorno presionado en el campo de contraseña.

```
void on_contrasenaLineEdit_returnPressed()
```

Maneja el evento de retorno presionado en el campo de correo electrónico.

```
void on_correoLineEdit_returnPressed()
```

Maneja el evento de retorno presionado en el campo de apellidos.

```
void on_apellidosLineEdit_returnPressed()
```

Maneja el evento de retorno presionado en el campo de nombres.

```
void on_nombresLineEdit_returnPressed()
```

Maneja el evento de clic en el botón de regresar.

```
void on_backButton_clicked()
```

Clase: UserWindow

Constructor de la clase UserWindow. Inicializa la ventana del usuario de la aplicación.

```
*UserWindow(QWidget parent = nullptr)
```

Destructor de la clase UserWindow. Libera recursos.

```
~UserWindow()
```

Métodos Privados

Maneja el evento de cierre de sesión.

```
void on_actionCerrar_Sesion_triggered()
```

Maneja el evento de clic en el botón de buscar.

```
void on_btnBuscar_clicked()
```

Maneja el evento de retorno presionado en el campo de búsqueda.

```
void on_inputBuscar_returnPressed()
```

Maneja el evento de clic en el panel del administrador.

```
void on_actionPanel_Administrador_triggered()
```

Maneja el evento de clic en el botón de modificar datos.

```
void on_btnModificarDatos_clicked()
```

Maneja el evento de clic en el botón de guardar modificaciones.

```
void on_guardarModificacionButton_clicked()
```

Maneja el evento de retorno presionado en el campo de nombres.

```
void on_nombresLogLineEdit_returnPressed()
```

Maneja el evento de retorno presionado en el campo de apellidos.

```
void on_apellidosLogLineEdit_returnPressed()
```

Maneja el evento de retorno presionado en el campo de correo electrónico.

```
void on_correoLogLineEdit_returnPressed()
```

Maneja el evento de retorno presionado en el campo de fecha de nacimiento.

```
void on_fechaDeNacimientoLogLineEdit_returnPressed()
```

Maneja el evento de retorno presionado en el campo de contraseña.

```
void on_contrasenaLogLineEdit_returnPressed()
```

Maneja el evento de clic en el botón para generar reportes.

```
void on_pushButton_generar_reportes_clicked()
```

Maneja el evento de clic en el botón para generar un reporte en BST.

```
void on_pushButton_reporte_generar_bst_clicked()
```

Maneja el evento de clic en el botón para abrir un reporte en BST.

```
void on_pushButton_reporte_abrir_bst_clicked()
```

Maneja el evento de clic en el botón para eliminar la cuenta.

```
void on_btnEliminarCuenta_clicked()
```

Maneja el evento de clic en el botón para crear un nuevo post.

```
void on_btn_nuevo_post_clicked()
```

Maneja el evento de clic en el botón para filtrar posts.

```
void on_btn_post_filtrar_clicked()
```

Maneja el cambio de pestaña en el tab widget.

```
void on_tabWidget_currentChanged(int index)
```