Universidad San Carlos de Guatemala

Facultad de Ingeniería

Escuela de Ciencias y Sistemas

Estructura de Datos

# MANUAL TECNICO

(FASE 3)

Victor Eduardo Franco Suret

202001767

26 de Octubre de 2024

## OBJETIVO DE ESTE MANUAL

El presente manual técnico tiene como finalidad describir aspectos técnicos y la forma en la que está estructurado, así como la interacción del usuario con este. Se muestra una fase 3 del proyecto general que es una aplicación de escritorio con una interfaz gráfica de usuario (GUI), lo que proporcionará una experiencia más intuitiva y accesible para los usuarios.

## DIRIGIDO

El uso de este manual es exclusivo para programadores o diseñadores que desean saber más sobre el funcionamiento técnico de la aplicación. Para la comprensión de este manual se debe tener conocimientos previos sobre el lenguaje de programación C++.

## CONOCIMIENTOS PREVIOS

Los conocimientos mínimos que deben tener las que utilizaran este manual son:

- Conocimientos sobre el Lenguaje C++

-Uso de Apuntadores, arboles, blockchain, árbol de merkle, grafos, compresión y decompresion de archivos.

-Conocimientos Generales en el uso de las librerías Graphviz y Nlohmann(para archivos json)

## ESPECIFICACIONES TECNICAS

Sistema Operativo: Windows 11

Lenguaje de Programación: c++.

IDE: Se utilizo Qt Creators

Tener instalado Graphviz , Nlomann y librería Sha256.

Siendo esta la Fase 3 del proyecto, se dará énfasis a las nuevas funcionalidades, así como las sustituciones de la fase anterior.

Vadmin referencia a la ventana del administrado el cual tiene acceso total para adición, edición y eliminación de datos.

Vuser referencia a la ventana de usuarios, en este es donde se concentra la mayoría de código siendo que es la que ejecute mas cantidad de operaciones.

Para ingresar al usuario administrador se toman las mismas credenciales:
Usuario: admin@gmail.com

Password: EDD2S2024

# Relaciones de Amistad

Propósito y Funcionalidad:

Escalabilidad y Eficiencia: La matriz de relaciones, utilizada inicialmente para representar las amistades entre los usuarios, presentaba problemas de duplicación de datos y un gran consumo de memoria, ya que cada usuario se almacenaba tanto en las filas como en las columnas, lo cual generaba un desperdicio considerable de recursos cuando se añadían más usuarios. Al cambiar a un grafo no dirigido utilizando una lista de adyacencia, se logró una estructura mucho más escalable y eficiente para manejar redes grandes con millones de usuarios.

Reducción de Complejidad: La matriz de adyacencia tiene una complejidad espacial de O(n2 ), donde n es el número de usuarios. Esta complejidad es ineficiente si la mayoría de los usuarios no tienen muchas conexiones. En contraste, un grafo con una lista de adyacencia tiene una complejidad espacial de O(V+E), donde V es el número de vértices (usuarios) y E es el número de aristas (relaciones). Esta representación reduce drásticamente el uso de memoria en redes grandes y dispersas.

Nodos Representando Usuarios: Cada nodo del grafo representa a un usuario. En la lista de adyacencia, los nodos contienen la información básica del usuario (como nombre y correo electrónico).

Aristas Representando Amistades: Cada arista en el grafo representa una amistad entre dos usuarios. La conexión es no dirigida (bidireccional), lo que significa que si "Usuario A" es amigo de "Usuario B", ambos usuarios están conectados de forma simétrica.

Eficiencia en la Búsqueda y Adición de Amistades:

Cuando un usuario envía una solicitud de amistad y esta es aceptada, se añade una arista entre los nodos correspondientes de los dos usuarios.

Para buscar a los amigos de un usuario, simplemente se navega por la lista de vecinos (adyacencia) del nodo de ese usuario, lo que facilita la búsqueda de amigos de manera eficiente.

Compatibilidad con Algoritmos de Recorrido: El uso de grafos también facilita la implementación de algoritmos de búsqueda (como BFS o DFS) para encontrar usuarios a ciertas "distancias", lo cual resulta útil para implementar funcionalidades como sugerencias de amistad.

Características Clave:

Lista de Adyacencia:

La estructura del grafo se implementó mediante una lista de adyacencia en lugar de una matriz. Cada usuario tiene una lista de sus amigos directos. Esta estructura almacena solo las conexiones existentes, reduciendo significativamente el uso de memoria.

En lugar de usar QList, la estructura se implementó utilizando una lista personalizada donde cada nodo tiene punteros a sus vecinos (amigos), siguiendo una implementación más cercana al hardware y menos dependiente de bibliotecas de alto nivel.

Inserción y Eliminación de Amigos:

Agregar Usuario: Cuando se crea un nuevo usuario en la aplicación, se añade un nuevo nodo en la lista de adyacencia.

Agregar Amistad: Para agregar una nueva amistad, se actualiza la lista de adyacencia de ambos usuarios, añadiendo cada uno al listado de amigos del otro.

Eliminar Amistad o Usuario: Al eliminar un usuario, se elimina el nodo correspondiente y se remueven todas las referencias a ese usuario en las listas de amigos de otros usuarios.

Sugerencias de Amistad y Exploración de Distancias:

Con la lista de adyacencia, se pueden implementar sugerencias de amistad a partir de usuarios que están a una distancia de dos saltos (es decir, amigos de amigos).

Se puede ponderar la relevancia de las sugerencias en base a cuántos amigos en común tiene un usuario con otro, lo cual se facilita recorriendo las listas de amigos de los vecinos de un usuario específico.

Grafo No Dirigido:

Las conexiones de amistad son bidireccionales, lo que significa que si "Usuario A" es amigo de "Usuario B", ambos estarán presentes en las listas de amigos del otro. Esto permite reflejar la naturaleza simétrica de las relaciones de amistad.

# Blockchain

Propósito y Funcionalidad:

Almacenamiento Seguro y Descentralizado: La blockchain se utiliza para almacenar la información de publicaciones y comentarios de una manera segura y confiable, haciendo que estos registros sean inmutables y resistentes a manipulaciones.

Auditoría y Verificación de Datos: La blockchain facilita la auditoría de las actividades dentro de la aplicación, permitiendo al administrador verificar que los datos de publicaciones y comentarios no hayan sido alterados sin autorización. Cualquier manipulación sería evidente debido a la naturaleza inmutable de la blockchain.

Registro de Publicaciones y Comentarios: Cada publicación realizada por un usuario, así como sus comentarios, se registra en bloques de la blockchain. Cada bloque contiene información relacionada, como el contenido, autor, y un timestamp (fecha y hora).

Estructura de Merkle Tree: Para garantizar la integridad de los datos dentro de un bloque, se utiliza un árbol de Merkle. Los árboles de Merkle permiten verificar si una transacción o comentario específico está presente en un bloque, sin necesidad de revisar todos los datos de la blockchain. Esto asegura la integridad y reduce la necesidad de espacio y procesamiento.

Prevención de Manipulación de Datos: La blockchain usa un algoritmo de hashing (como SHA-256) para proteger cada bloque. Cada bloque de la cadena contiene un hash que se genera a partir del contenido del bloque anterior. Si alguien intentara modificar el contenido de un bloque, los hashes cambiarían, haciendo que la manipulación sea fácilmente detectable.

Manejo de Datos Sensibles: La blockchain no solo almacena información de publicaciones, sino que también protege datos sensibles como contraseñas. Las contraseñas se comprimen y cifran (utilizando compresión de Huffman) antes de ser almacenadas.

Características Clave:

Inmutabilidad de los Datos:

Una vez que los datos (publicaciones o comentarios) se registran en un bloque y se agrega a la cadena, no pueden ser alterados sin afectar todos los bloques siguientes. Esto asegura la inmutabilidad, protegiendo la integridad de la información almacenada.

Integridad con Árbol de Merkle:

Cada bloque contiene un árbol de Merkle que permite verificar rápidamente si ciertos datos están contenidos dentro del bloque. Esto garantiza que la estructura interna de cada bloque no sea manipulada y facilita la validación de datos sin recorrer toda la cadena.

Cifrado Seguro de Datos Sensibles:

Se utiliza SHA-256 para el hashing y la seguridad general de los bloques. Además, para la protección específica de datos sensibles, como contraseñas de los usuarios, se implementa un sistema de compresión de Huffman para reducir la información y cifrarla antes de agregarla a la blockchain. Esto ofrece una doble capa de seguridad.

Hashing y Vinculación de Bloques:

Cada bloque contiene un hash que enlaza con el bloque anterior, formando una cadena continua. Este enfoque asegura que si se altera un bloque, los hashes de los bloques subsiguientes también cambian, indicando manipulación. Esto es lo que hace la blockchain segura contra intentos de fraude o corrupción de datos.

Proceso de Creación de Bloques:

Los datos de publicaciones y comentarios se agrupan en bloques que se crean periódicamente o cuando hay suficientes transacciones para llenar un bloque. Cada bloque nuevo incluye:

Hash del bloque anterior para mantener la conexión segura.

Timestamp, que indica la fecha y hora de creación del bloque.

Árbol de Merkle para mantener la integridad de las transacciones.