PROYECTO BLOCKCHAIN

GRUPO 5

- HERENCIA GUERRA, JUAN
- LEONARDO JULIAN, CARLOS
- ROJAS BARRAZA, CLUNY
- SILVA CAJUSOL, MIGUEL

Descripción del caso de estudio planteado por el grupo

Problema

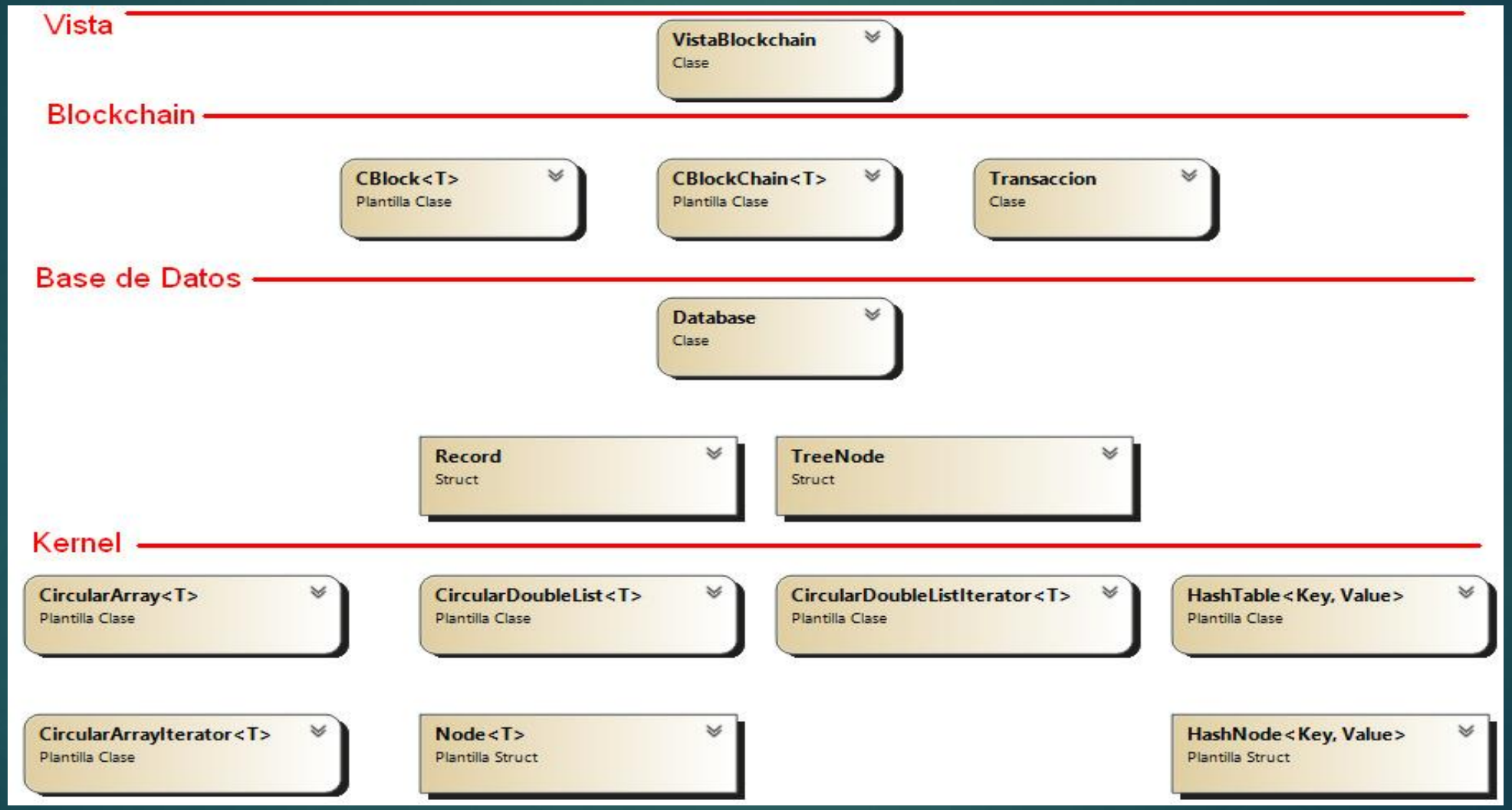
- En las entidades bancarias y financieras, la seguridad y privacidad de los datos son aspectos críticos para las transacciones bancarias. Los sistemas tradicionales de almacenamiento y búsqueda de datos pueden presentar vulnerabilidades, y el acceso no autorizado a la información confidencial puede tener consecuencias graves, como el robo de identidad o el fraude financiero. Existe la necesidad de desarrollar una solución confiable y segura que garantice la protección de los datos transaccionales y, al mismo tiempo, permita búsquedas eficientes para su procesamiento. Debido a esa necesidad se propone la implementación de una solución basada en blockchain.

Objetivo

- Desarrollar una aplicación transaccional basada en blockchain y que sea segura y eficiente para el almacenamiento de datos de transferencias bancarias.

SOLUCION DEL PROYECTO

Arquitectura

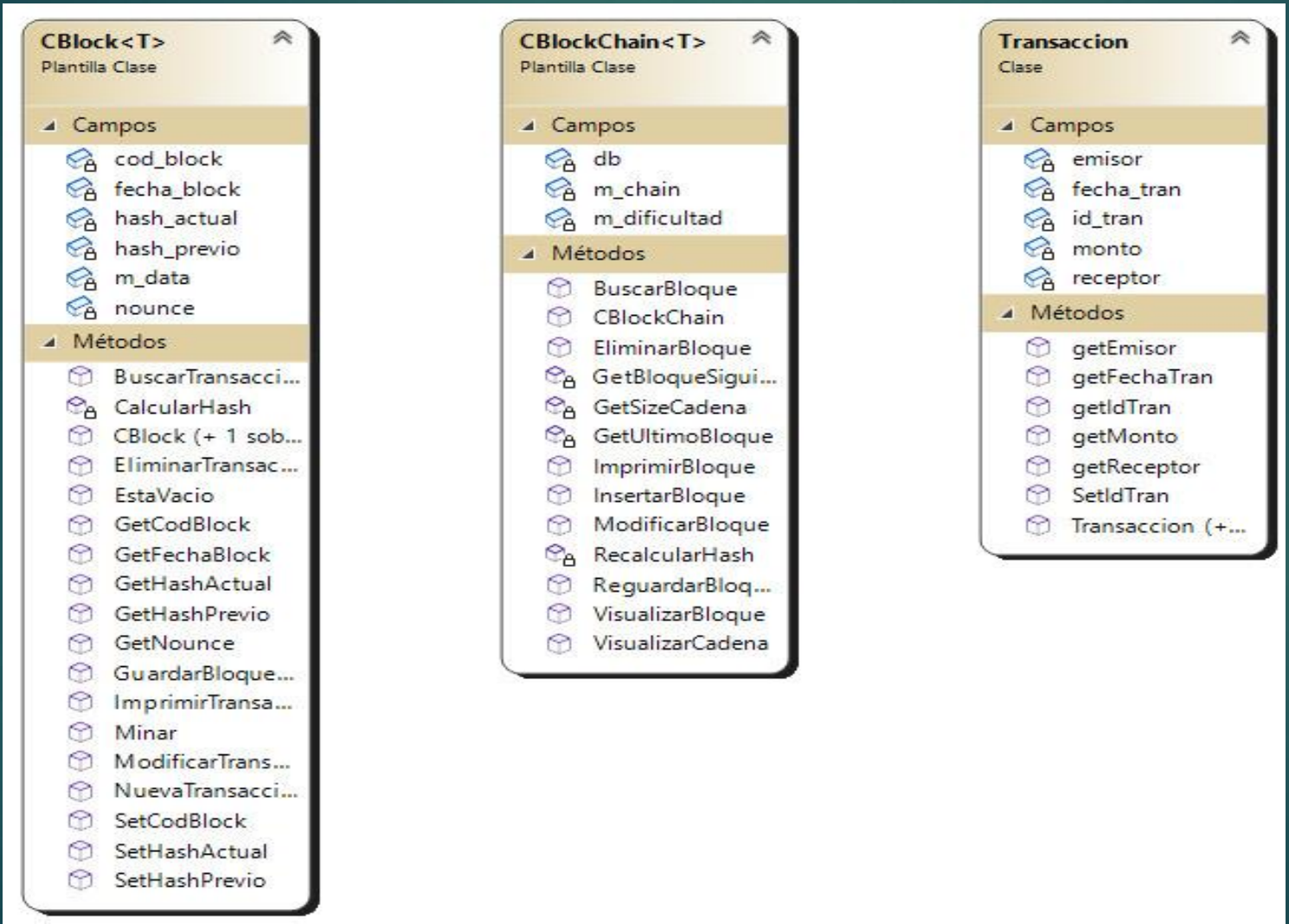


Capa Vista

1. **Clase VistaBlockchain.h**

Clase	Metodo	Descripcion
VistaBlockchain.h	void menuPrincipal(CBlockChain<CBlock<Transaccion>>& cadena);	Metodo que lista las funcionalidades del proyecto
VistaBlockchain.h	void opcionAgregarBloque(CBlockChain<CBlock<Transaccion>>& cadena);	Metodo que llama a los metodos para Insertar el bloque e insertar transacciones
VistaBlockchain.h	Transaccion ingresarNuevaTransaccion();	Metodo que solicita los datos de cada transaccion que se desea ingresar en el bloque.
VistaBlockchain.h	void mostrarTransaccion(Transaccion trx);	Metodo para mostrar los datos de la transaccion
VistaBlockchain.h	void opcionModificarBloque(CBlockChain<CBlock<Transaccion>>& cadena);	Metodo con las subopciones de la opcion Modificar Bloque
VistaBlockchain.h	void supOpcionNuevaTransaccionPorBloque(CBlockChain<CBlock<Transaccion>>& cadena);	Metodo que llama a los metodos para Insertar nueva transaccion a un bloque ya existente
VistaBlockchain.h	void subOpcionModificarTransaccion(CBlockChain<CBlock<Transaccion>>& cadena);	Metodo que llama a los metodos para modificar una transaccion de un bloque ya existente
VistaBlockchain.h	void subOpcionEliminarTransaccion(CBlockChain<CBlock<Transaccion>>& cadena);	Metodo que llama a los metodos para eliminar una transaccion determinada de un bloque ya existente
VistaBlockchain.h	void opcionEliminarBloque();	Metodo que llama a los metodos para eliminar un bloque ya existente
VistaBlockchain.h	void opcionCargarDesdeArchivo();	Metodo que llama a los metodos para cargar bloques de un archivo csv
VistaBlockchain.h	void opcionGuardarEnArchivo();	Metodo que llama a los metodos para guardar los bloques generados en un archivo csv
VistaBlockchain.h	void opcionVisualizar(CBlockChain<CBlock<Transaccion>>& cadena);	Metodo con las subopciones de la opcion Visualizar
VistaBlockchain.h	void subOpcionVisualizarBloque(CBlockChain<CBlock<Transaccion>>& cadena);	Metodo que llama al metodo para visualizar un bloque determinado
VistaBlockchain.h	void opcionBusqueda();	Metodo con las subopciones de la opcion Busqueda y listado

Capa Blockchain



Capa Blockchain

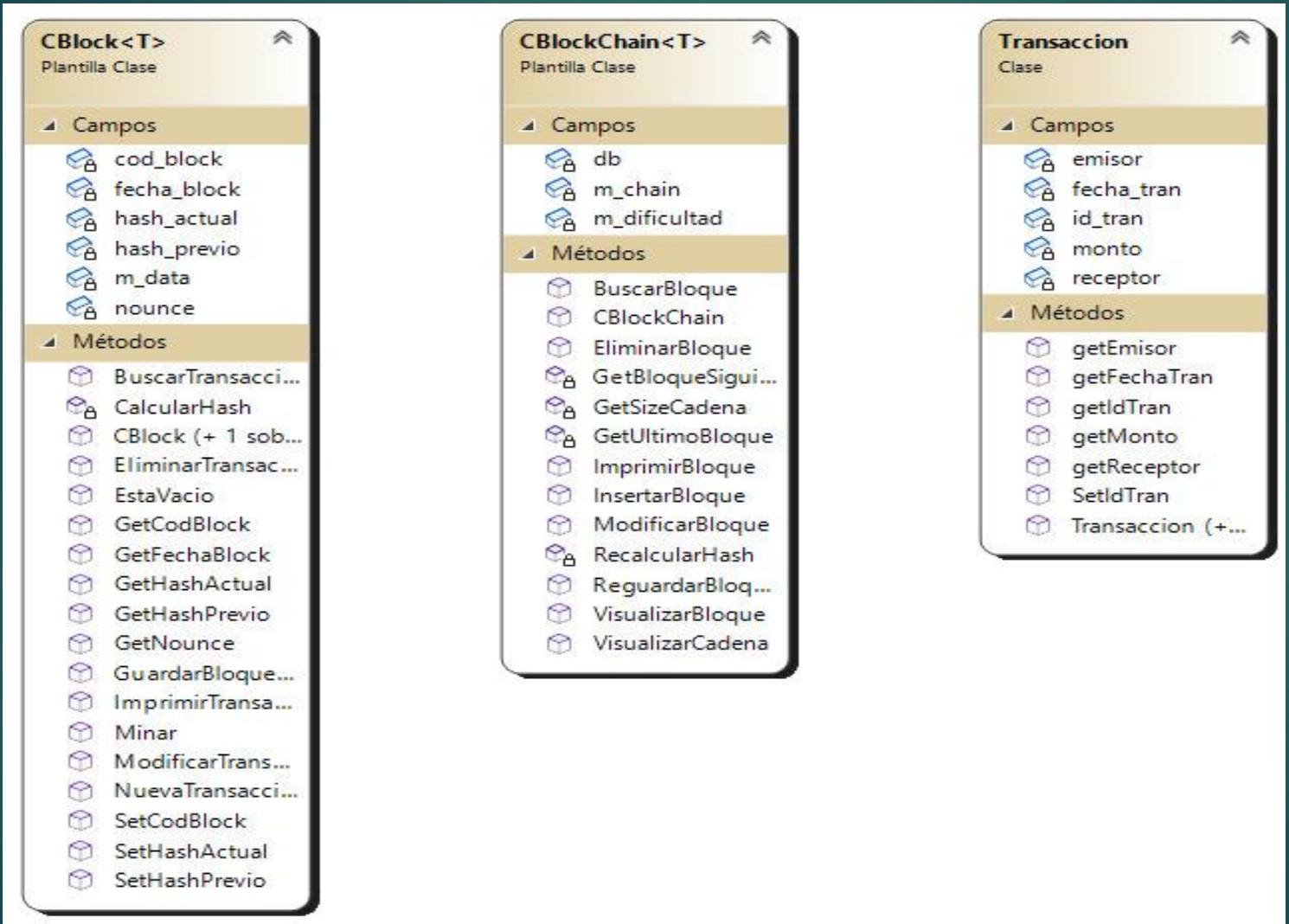
1. **Clase Blockchain.h**

Metodo	Descripcion	Entrada	Salida
void InsertarBloque(T& nuevoBloque)	Metodo para agregar un nuevo bloque al blockchain (cadena de bloques). Se le asigna un codigo y el hash previo, se invoca al metodo Minar() para el calculo del hash actual, usamos el metodo push_back(T data) de nuestra estructura de datos Lista Circular Doblemente enlazada y finalmente guardamos el bloque en BD.	Bloque Nuevo	Agregar el nuevo bloque
T BuscarBloque(int IdBloque);	Metodo para buscar el bloque por el codigo.	Codigo del Bloque	Bloque buscado
void ModificarBloque(T nuevoBloque);	Metodo que actualiza la informacion del bloque y realiza lel recalculo del hash del bloque y sus subsiguientes.	Bloque modificado	Recalculo de los hash del bloque y los bloques dependientes
void VisualizarBloque(int IdBloque);	Metodo para visualizar la informacion de un bloque en especifico.	Codigo del Bloque	Hash previo, hash actual, nonce, fecha bloque y la informacion de cada una de sus transacciones
void VisualizarCadena();	Metodo para visualizar la informacion de todos los bloques.		El listado de bloques con su respectiva informacion
void RecalcularHash(T bloque)	Metodo que realiza el recalculo del bloque y los bloques subsiguientes para actualizar el hash actual y el hash previo de cada bloque.	Bloque	Hash actualizado del bloque y los bloques subsiguientes
CBlockChain()	Metodo constructor que asigna la dificultad de la prueba de trabajo y genera el bloque genesis.	Dificultad de la prueba de trabajo	Bloque genesis agregado al blockchain

Proof of work

```
// Cosntructor
template<class T>
CBlockChain<T>::CBlockChain() {
	this->m_dificultad = DIFICULTAD_PRUEBA_DE_TRABAJO;
	T BloqueGenesis;
	BloqueGenesis.Minar(this->m_dificultad);
	this->m_chain.push_back(BloqueGenesis); // El BlockChain nace con el
bloque Génesis
	}


// Metodo Minar
template<class T>
void CBlock<T>::Minar(int difficulty) {
	char* cstr = new char[difficulty + 1];

	for (uint32_t i = 0; i < difficulty; ++i) {
		cstr[i] = '0';
	}
	cstr[difficulty] = '\0';

	string str(cstr);

	while (true) {
		this->nounce++;
		this->hash_actual = CalcularHash();

		if (this->hash_actual.substr(0, difficulty) == str) {
			break;
		}
	}
	if(this->cod_block != 0)
		cout << "Bloque " << this->cod_block << " minado: " << this-
>hash_actual << endl;
		}
```

Capa Blockchain

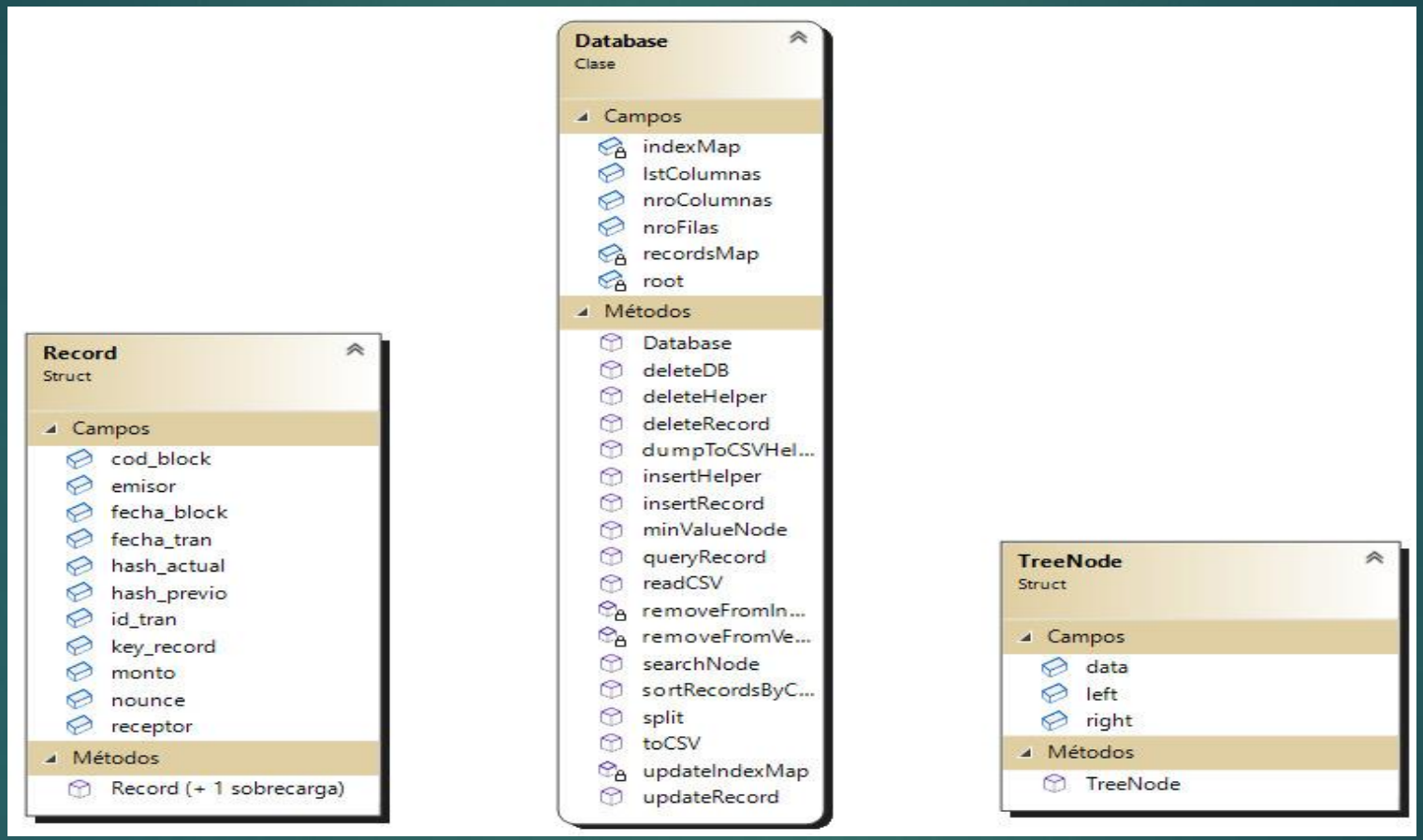


Capa Blockchain

1. **Clase Block.h**

Método	Descripción
string CalcularHash()	Método que verifica si el array circular donde se alojan los datos está vacío o no, y si contiene datos, hace un recorrido con un iterador para obtener los datos de la transacción y hace uso del método SHA256 para generar un hash.
void Minar(int difficulty)	Método para minar el bloque mediante un bucle que incrementa el valor de nonce y recalcula el valor del hash a partir de este con el método CalcularHash().
void NuevaTransaccion(T& data)	Método para verificar si el tamaño del array circular que almacena los datos no sobrepasa el máximo de transacciones por bloque y partir de esto hace uso del método push_back para agregar una nueva transacción.
void ModificarTransaccion(T data)	Método para modificar una transacción de acuerdo a la búsqueda de su id con un recorrido que se hace mediante una iteración en el array circular m_data
bool BuscarTransaccion(int IdTran)	Método para buscar una transacción, de acuerdo a su id, mediante un recorrido por el array circular m_data con una iteración.

Capa BD


Record
Struct
Campos
cod_block
emisor
fecha_block
fecha_tran
hash_actual
hash_previo
id_tran
key_record
monto
nounce
receptor
Métodos
Record (+ 1 sobrecarga)
Database
Clase
Campos
indexMap
lstColumnas
nroColumnas
nroFilas
recordsMap
root
Métodos
Database
deleteDB
deleteHelper
deleteRecord
dumpToCSVHel...
insertHelper
insertRecord
minValueNode
queryRecord
readCSV
removeFromIn...
removeFromVe...
searchNode
sortRecordsByC...
split
toCSV
updateIndexMap
updateRecord
TreeNode
Struct
Campos
data
left
right
Métodos
TreeNode

Capa BD

Método	Descripción
int gen_key_record(int cod_block, int id_tran)	Método para la generación de una clave única para cada registro. Retorna un código de tipo entero basado en la multiplicación del código o id del bloque por 1 millón, y se suma con el id de la transacción
std::string* split(const std::string& line, char delimiter)	
std::vector<int> calculateColumnWidths()	Método que calcula el número de columnas de un bloque
bool readCSV(const std::string& filename, char delimiter)	Método para la lectura y guardado de los datos del archivo csv
void dumpToCSVHelper(TreeNode* node, std::ofstream& file)	Método para separar los datos por columna de cada registro almacenado en el archivo csv
bool toCSV(std::string& nameFile)	Médodo donde se llenan los campos de un archivo csv a ser creado.
void insertRecord(const Record& record)	Método para guardar un registro en una estructura de tipo árbol binario de búsqueda
TreeNode* insertHelper(TreeNode* node, TreeNode* newNode)	Método para insertar un nodo al árbol binario de búsqueda, este nuevo nodo se agregará a la izquierda o derecha del primer nodo ingresado dependiendo de si es menor o mayor que esta. Si el nuevo nodo es nulo, se inserta en la raíz.
void deleteRecord(int id_block, int id_tran = -1)	Método para eliminar un registro o transacción de la base de la base de datos.
void deleteDB(Treenode* node)	Método para borrar toda la base de datos.
TreeNode* deleteHelper(TreeNode* node, TreeNode* deleteNode)	Método para eliminar el nodo ubicado a la izquierda o a la derecha dependiendo de la condición cumplida.
TreeNode* minValueNode(TreeNode* node)	Método para hallar el hijo de menor valor de un nodo
void updateRecord(const Record& updatedRecord)	Método que retorna un valor 1 si se actualiza el registro y cero en caso opuesto.
Record queryRecord(int id_block, int id_tran = -1)	Método que retorna la data de una transacción
TreeNode* searchNode(int id)	Método que busca un nodo por su id, si este no se encuentra, se devuelve un valor nulo.
void sortRecordsByColumn(const std::string& column)	Método para ordenar los datos por alguna columna
void updateIndexMap(TreeNode* node)	Método para actualizar los campos almacenados en la base de datos.
void removeFromIndexMap(TreeNode* node)	Método para eliminar los campos almacenados en la base de datos-
void removeFromVector(std::vector<TreeNode*>& vec, TreeNode* node)	Método que elimina un vector de acuerdo al nodo ingresado.

EJECUCIÓN DEL DEMO

GRACIAS!!