Proyecto Laboratorio Computacional 3
Blockchain de “Libro Diario”

Integrantes Grupo:

- Flores Abel

Descripción:

El sistema permite la persistencia de un libro diario en forma de bloque en una blockchain. Un libro diario estara compuesto por una serie de asientos los cuales a su vez tendran una serie de movimientos. Un movimiento consiste de un monto en Debe o en Haber y una cuenta a la cual corresponde el movimiento.

Código Fuente:

Las clases que componen la estructura de la Blockchain son Block.cs y BlockChain.cs.

Block.cs

```
[Serializable]
23 referencias
internal class Block
{

    5 referencias
    public int Index
        { get; set; }
    2 referencias
    public DateTime TimeStamp { get; set; }

    5 referencias
    public String PreviousHash { get; set; }

    7 referencias
    public String Hash { get; set; }

    3 referencias
    public LibroDiario Data { get; set; }

    2 referencias
    public Block(DateTime timeStamp, String previousHash, LibroDiario data)
    {
        Index = 0;
        TimeStamp = timeStamp;
        PreviousHash = previousHash;
        Data = data;
        Hash = CalculateHash();
    }

    4 referencias
    public String CalculateHash()
    {
        SHA256 sHA256 = SHA256.Create();
        byte[] inputBytes = Encoding.ASCII.GetBytes($"{ TimeStamp}-{ PreviousHash ?? ""}-{ Data}");
        byte[] outputBytes = sHA256.ComputeHash(inputBytes);
        return Convert.ToBase64String(outputBytes);
    }
}
```

Esta clase representa un bloque que tendra la BlockChain. Contiene un atributo que representa la Hash del bloque anterior y su propio Hash el cual se calculó teniendo en cuenta el hash del bloque anterior y la data que está guardando. De esta forma es imposible insertar un bloque “maligno” sin tener que modificar todos los bloques que le siguen a este.

BlockChain.cs

```
[Serializable]
5 referencias
internal class BlockChain
{
    16 referencias
    public List<Block> Chain { get; set; }

    private const string BASE_PATH = "/bloques/";
    private const string GENESIS_NAME = "GENESIS.bin";

    2 referencias
    public BlockChain()
    {
        int cantFiles = Directory.GetFiles(@"/bloques").Length;
        if (File.Exists(BASE_PATH + GENESIS_NAME))
        {
            Block genesisBlock = BinarySerialization.ReadFromBinaryFile<Block>(BASE_PATH + GENESIS_NAME);
            LoadBlockChainFromFileSystem(genesisBlock);
        }
        else
        {

            InitializeChain();
            AddGenesisBlock();
        }

        if(cantFiles > Chain.Count) throw new InvalidBlockException("Se perdio un bloque.");

        foreach (Block block in Chain)
        {
            Console.WriteLine(JsonSerializer.Serialize<Block>(block));
        }
    }
```

Esta clase contiene una lista de bloques la cual será la representacion de la blockchain en memoria. Al inicializar una instancia esta se fijará si existen archivos de bloque persistidos, en caso de que no existan creará el bloque genesis y lo guardará en memoria.

```
public void LoadBlockChainFromFileSystem(Block genesisBlock)
{
    int index = genesisBlock.Index;
    Chain = new List<Block>();
    Chain.Add(genesisBlock);

    while(true)
    {
        index++;
        string nextPath = @BASE_PATH+(index)+".bin";
        if (File.Exists(nextPath))
        {
            Block loadedBlock = BinarySerialization.ReadFromBinaryFile<Block>(nextPath);

            if (!IsValidBlock(index, loadedBlock)) throw new InvalidBlockException("EL BLOQUE Nro " + index + " es invalido.");

            Chain.Add(loadedBlock);
        }
        else
        {
            break;
        }
    }
}
```

Este metodo cargará en memoria los bloques persistidos. Estos se cargan segun el nombre del archivo el cual deberia tener como nombre el índice al que corresponde el bloque.

```
public bool IsValidBlock(int index, Block currentBlock)
{
    Block previousBlock = Chain[index - 1];

    if (currentBlock.Hash != currentBlock.CalculateHash() || currentBlock.PreviousHash != previousBlock.Hash)
    {
        return false;
    }

    return true;
}
```

Este metodo valida un bloque, se fija por un lado que el hash que indica el bloque se coincida con el hash que produce el bloque si lo paso por el metodo de cálculo de hash (esto es para evitar poder modificar el contenido del bloque sin cambiar su hash), luego también calcula que el que indica como hash del bloque anterior sea efectivamente el hash del bloque anterior.

```
public void AddBlock(Block block)
{
    Block latestBlock = GetLatestBlock();
    block.Index = latestBlock.Index + 1;
    block.PreviousHash = latestBlock.Hash;
    block.Hash = block.CalculateHash();

    BinarySerialization.WriteToBinaryFile(BASE_PATH + block.Index + ".bin", block);

    Chain.Add(block);
}
```

Este metodo agrega un bloque a la blockchain tanto en memoria con en disco, serializandolo y persisitendolo.

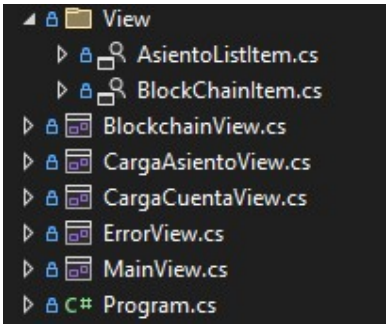
BinarySerialization.cs

```
public static class BinarySerialization
{
    2 referencias
    public static void WriteToBinaryFile<T>(string filePath, T objectToWrite, bool append = false)
    {
        using (Stream stream = File.Open(filePath, append ? FileMode.Append : FileMode.Create))
        {
            stream.Seek(0, SeekOrigin.Begin);
            var binaryFormatter = new System.Runtime.Serialization.Formatters.Binary.BinaryFormatter();
            binaryFormatter.Serialize(stream, objectToWrite);
        }
    }

    2 referencias
    public static T ReadFromBinaryFile<T>(string filePath)
    {
        using (Stream stream = File.Open(filePath, FileMode.Open))
        {
            stream.Seek(0, SeekOrigin.Begin);
            var binaryFormatter = new System.Runtime.Serialization.Formatters.Binary.BinaryFormatter();
            return (T)binaryFormatter.Deserialize(stream);
        }
    }
}
```

Esta es la clase que serializara los bloques y los guardará/leera en disco.

Clases de Vistas:



Estas son las siguientes clases de vistas, de las mismas quizas la mas importante es la que confirmar el LibroDiario y lo manda a persistir en la BlockChain.

```
1 referencia
private void btnConfirmarLibro_Click(object sender, EventArgs e)
{
    foreach (Asiento asiento in AsientoRepository.Consultar())
    {
        Console.WriteLine(asiento.Concepto);
    }

    LibroDiario libroDiario = new LibroDiario()
    {
        fecha = DateTime.Now,
        asientos = AsientoRepository.Consultar()
    };

    Console.WriteLine(libroDiario.asientos.Count);

    LibroDiarioRepository.GuardarEnBlockchain(libroDiario);

    AsientoRepository.Vaciar();

    flowLayoutPanelAsientos.Controls.Clear();
}
```

Se le asigna la fecha del día al libro y se recuperan los asientos que están cargados en memoria, luego se manda el LibroDiario a guardar en BlockChain y se vacian los asientos que estaban actualmente en memoria para dar lugar a un nuevo libro que queramos cargar.

El metodo LibroDiarioRepository.GuardarEnBlockchain:

```
1 referencia
public static bool GuardarEnBlockchain(LibroDiario entity)
{
    Program.mainBlockchain.AddBlock(new Block(DateTime.Now, null, entity));

    return true;
}
```

El mismo crea un Block con el librodiario como Data y manda a persistir a la clase Blockchain.