**Tutorial – MongoDB with BlockChain**

1. Implement the example demonstrated in the class.
2. Open VS 2022, Create a c# console project

A screenshot of a computer

Description automatically generated

1. Save your project in a location and pick the latest standard support framework.

A screenshot of a computer

Description automatically generated

1. Create 3 new cs classes, Block, BlockChain, Progrm. Insert the codes give above .
2. Include MongoDB Drive from NuGet

A screenshot of a computer program

Description automatically generated

1. Run your application, you will get a valid flag of the blockchain create.
2. Now log into MongoDB, explore MongoDB Blockchain Database create, Observe the data

A screenshot of a chat

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

1. Add more block to the chain
2. Implement a new C# programme where the block data is saving as a MongoDB Document instead of a string (demonstrated in the class, codes is provided)
3. Compare the difference between the data stored in the previous version and the 2nd version of the BlockChain programme

**Theoretical Question**

1. How does the blockchain technology work ? Illustrate with a diagram and an explanation
2. (5 marks)
3. Compare a blockchain with a Databases? (5 marks)
4. Give two examples how blockchain can be used in Retail, Digital Forensic? (10 marks)

Appendix :

**Prerequisites:**

* Basic knowledge of C# programming
* Visual Studio 2022 installed
* MongoDB installed and running on your system
* MongoDB .NET driver (NuGet package) installed in your project - ‘dotnet add package MongoDB.Driver’

**Step 1: Set Up MongoDB**

1. Install MongoDB: Follow the installation instructions for your operating system from the official MongoDB documentation.

**Step 2: Create C# Project**

1. Open Visual Studio 2022 and create a new C# Console Application project.

**Step 3: Create Block Class**

1. Create a **Block** class to represent each block in the blockchain:

public class Block

{

public int Index { get; set; }

public string PreviousHash { get; set; }

public long Timestamp { get; set; }

public string Data { get; set; }

public string Hash { get; set; }

}

**Step 4: Create Blockchain Class**

1. Create a **Blockchain** class to manage the blockchain:

using System;

using System.Collections.Generic;

using MongoDB.Bson;

using MongoDB.Driver;

using System.Security.Cryptography;

using System.Text;

public class Blockchain

{

private List<Block> chain;

private IMongoCollection<Block> blocksCollection;

public Blockchain()

{

chain = new List<Block>();

var mongoClient = new MongoClient("mongodb://localhost:27017");

var database = mongoClient.GetDatabase("BlockchainDB");

blocksCollection = database.GetCollection<Block>("Blocks");

Block genesisBlock = CreateGenesisBlock();

blocksCollection.InsertOne(genesisBlock);

chain.Add(genesisBlock);

}

private Block CreateGenesisBlock()

{

return new Block

{

Index = 0,

PreviousHash = "0",

Timestamp = DateTimeOffset.Now.ToUnixTimeSeconds(),

Data = "Genesis Block",

Hash = CalculateHash("0")

};

}

private string CalculateHash(string input)

{

using (SHA256 sha256 = SHA256.Create())

{

byte[] inputBytes = Encoding.UTF8.GetBytes(input);

byte[] hashBytes = sha256.ComputeHash(inputBytes);

StringBuilder builder = new StringBuilder();

foreach (byte b in hashBytes)

{

builder.Append(b.ToString("x2"));

}

return builder.ToString();

}

}

public void AddBlock(string data)

{

Block previousBlock = chain[^1];

Block newBlock = new Block

{

Index = previousBlock.Index + 1,

PreviousHash = previousBlock.Hash,

Timestamp = DateTimeOffset.Now.ToUnixTimeSeconds(),

Data = data,

Hash = CalculateHash(previousBlock.Hash + data)

};

blocksCollection.InsertOne(newBlock);

chain.Add(newBlock);

}

public bool IsChainValid()

{

for (int i = 1; i < chain.Count; i++)

{

Block currentBlock = chain[i];

Block previousBlock = chain[i - 1];

if (currentBlock.Hash != CalculateHash(previousBlock.Hash + currentBlock.Data))

{

return false;

}

}

return true;

}

}

}

**Step 5: Using the Blockchain**

1. In your **Program.cs** file, create an instance of the **Blockchain** class and add blocks to the blockchain:

}

class Program

{

static void Main(string[] args)

{

Blockchain blockchain = new Blockchain();

blockchain.AddBlock("Transaction Data 1");

blockchain.AddBlock("Transaction Data 2");

// Add more blocks as needed

bool isValid = blockchain.IsChainValid();

Console.WriteLine($"Is blockchain valid? {isValid}");

}

}

Remember that this tutorial provides a basic introduction to building a blockchain using C# and MongoDB. In a real-world scenario, you'd need to implement more features like proof-of-work, consensus algorithms, and network communication for a fully functional blockchain system. Also, ensure you have the MongoDB .NET driver installed in your project using NuGet package manager for MongoDB interactions to work properly.

**Data property - can it be a MongoDB document?**

Here's an example of how you could modify the **Block** class to use a MongoDB document as the **Data** property:

In this modified version of the **Block** class:

1. The **Data** property is now of type **BsonDocument**. **BsonDocument** is a class provided by the MongoDB driver to represent a document in the MongoDB format.
2. The **[BsonRepresentation(BsonType.Document)]** attribute is applied to the **Data** property. This attribute tells the MongoDB driver that the **Data** property should be treated as a MongoDB document when saving and retrieving from the database.

Using a **BsonDocument** allows you to represent complex data structures and leverage MongoDB's capabilities for querying and working with nested data. However, keep in mind that this might add complexity to your blockchain implementation, so consider whether it aligns with your use case and objectives.

using MongoDB.Bson;

using MongoDB.Bson.Serialization.Attributes;

public class Block

{

public int Index { get; set; }

public string PreviousHash { get; set; }

public long Timestamp { get; set; }

[BsonRepresentation(BsonType.Document)] // Specify MongoDB representation

public BsonDocument Data { get; set; }

public string Hash { get; set; }

}

using MongoDB.Bson;

using MongoDB.Driver;

using System;

using System.Collections.Generic;

using System.Security.Cryptography;

using System.Text;

public class Blockchain

{

private List<Block> chain;

private IMongoCollection<Block> blocksCollection;

public Blockchain()

{

chain = new List<Block>();

var mongoClient = new MongoClient("mongodb://localhost:27017");

var database = mongoClient.GetDatabase("BlockchainDB");

blocksCollection = database.GetCollection<Block>("Blocks");

Block genesisBlock = CreateGenesisBlock();

blocksCollection.InsertOne(genesisBlock);

chain.Add(genesisBlock);

}

private Block CreateGenesisBlock()

{

BsonDocument genesisData = new BsonDocument

{

{ "message", "Genesis Block" },

{ "initialValue", 100 }

};

return new Block

{

Index = 0,

PreviousHash = "0",

Timestamp = DateTimeOffset.Now.ToUnixTimeSeconds(),

Data = genesisData,

Hash = CalculateHash("0")

};

}

private string CalculateHash(string input)

{

using (SHA256 sha256 = SHA256.Create())

{

byte[] inputBytes = Encoding.UTF8.GetBytes(input);

byte[] hashBytes = sha256.ComputeHash(inputBytes);

StringBuilder builder = new StringBuilder();

foreach (byte b in hashBytes)

{

builder.Append(b.ToString("x2"));

}

return builder.ToString();

}

}

public void AddBlock(BsonDocument data)

{

Block previousBlock = chain[^1];

Block newBlock = new Block

{

Index = previousBlock.Index + 1,

PreviousHash = previousBlock.Hash,

Timestamp = DateTimeOffset.Now.ToUnixTimeSeconds(),

Data = data,

Hash = CalculateHash(previousBlock.Hash + data.ToString())

};

blocksCollection.InsertOne(newBlock);

chain.Add(newBlock);

}

public bool IsChainValid()

{

for (int i = 1; i < chain.Count; i++)

{

Block currentBlock = chain[i];

Block previousBlock = chain[i - 1];

if (currentBlock.Hash != CalculateHash(previousBlock.Hash + currentBlock.Data.ToString()))

{

return false;

}

}

return true;

}

}

class Program

{

static void Main(string[] args)

{

Blockchain blockchain = new Blockchain();

BsonDocument transactionData1 = new BsonDocument

{

{ "from", "Alice" },

{ "to", "Bob" },

{ "amount", 10 }

};

BsonDocument transactionData2 = new BsonDocument

{

{ "from", "Bob" },

{ "to", "Charlie" },

{ "amount", 5 }

};

blockchain.AddBlock(transactionData1);

blockchain.AddBlock(transactionData2);

bool isValid = blockchain.IsChainValid();

Console.WriteLine($"Is blockchain valid? {isValid}");

}

}

Top of Form

Bottom of Form