**🔗 Blockchain Core Functions Recap**

Each example includes:

* Block struct
* CalculateHash
* InsertBlock
* ChangeBlock
* ListBlocks
* VerifyChain

**1. Hospital Records Blockchain**

type Block struct {

    transactions []string // e.g., "Patient A - Diagnosis: Flu"

    prevPointer  \*Block

    prevHash     string

    currentHash  string

}

💡 **Example Transactions**: ["John Doe - Flu", "Jane Smith - X-ray"]

🔎 **Use Case**: Tamper-proof medical records history for patients.

**2. Bank Balance Ledger (Sender/Receiver Structure)**

type Block struct {

    Spender     map[string]int

    Receiver    map[string]int

    PrevPointer \*Block

    PrevHash    string

    CurrentHash string

}

**✅ Functions (Common in All)**

**🧮 CalculateHash**

func CalculateHash(block \*Block) string {

    data := block.prevHash

    for \_, transaction := range block.transactions {

        data += transaction

    }

    hash := sha256.Sum256([]byte(data))

    return hex.EncodeToString(hash[:])

}

**➕ InsertBlock**

func InsertBlock(transactions []string, chainHead \*Block) \*Block {

    newBlock := &Block{transactions, chainHead, "", ""}

    if chainHead != nil {

        newBlock.prevHash = chainHead.currentHash

    }

    newBlock.currentHash = CalculateHash(newBlock)

    return newBlock

}

**✏️ ChangeBlock**

func ChangeBlock(oldTrans, newTrans string, chainHead \*Block) {

    current := chainHead

    for current != nil {

        for i, t := range current.transactions {

            if t == oldTrans {

                current.transactions[i] = newTrans

                current.currentHash = CalculateHash(current)

                fmt.Println("Transaction changed.")

                return

            }

        }

        current = current.prevPointer

    }

    fmt.Println("Transaction not found.")

**📋 ListBlocks**

func ListBlocks(chainHead \*Block) {

    current := chainHead

    i := 1

    for current != nil {

        fmt.Printf("Block %d:\nTransactions: %v\nPrev Hash: %s\nCurr Hash: %s\n\n",

            i, current.transactions, current.prevHash, current.currentHash)

        current = current.prevPointer

        i++

    }

}

**🔍 VerifyChain**

func VerifyChain(chainHead \*Block) {

    current := chainHead

    for current != nil {

        if CalculateHash(current) != current.currentHash {

            fmt.Println("Blockchain compromised!")

            return

        }

        if current.prevPointer != nil && current.prevHash != current.prevPointer.currentHash {

            fmt.Println("Hash mismatch with previous block!")

            return

        }

        current = current.prevPointer

    }

    fmt.Println("Blockchain is valid.")

}

// Block represents a block in the blockchain

type Block struct {

    transactions []string

    prevPointer  \*Block

    prevHash     string

    currentHash  string

}

// CalculateHash computes the hash of a block: prevHash + data

func CalculateHash(block \*Block) string {

    data := block.prevHash

    for \_, transaction := range block.transactions {

        data += transaction

    }

    hash := sha256.Sum256([]byte(data))

    return hex.EncodeToString(hash[:])      //convert the array of 32 bytes into a string

}

// InsertBlock adds a new block to the blockchain

func InsertBlock(transactions []string, chainHead \*Block) \*Block {

    newBlock := &Block{

        transactions: transactions,

        prevPointer:  chainHead,

        prevHash:     "",

        currentHash:  "",

    }

    if chainHead != nil {

        newBlock.prevHash = chainHead.currentHash

    }

    newBlock.currentHash = CalculateHash(newBlock)

    return newBlock

}

// ChangeBlock modifies a transaction in a block

func ChangeBlock(oldTrans string, newTrans string, chainHead \*Block) {

    currentBlock := chainHead

    for currentBlock != nil {

        for i, transaction := range currentBlock.transactions {

            if transaction == oldTrans {

                currentBlock.transactions[i] = newTrans

                currentBlock.currentHash = CalculateHash(currentBlock)

                fmt.Println("Transaction changed successfully!")

                return

            }

        }

        currentBlock = currentBlock.prevPointer

    }

    fmt.Println("Transaction not found in the blockchain.")

}

// ListBlocks displays all blocks in the blockchain

func ListBlocks(chainHead \*Block) {

    currentBlock := chainHead

    blockNumber := 1

    for currentBlock != nil {

        fmt.Printf("Block %d:\n", blockNumber)

        fmt.Println("Transactions:", currentBlock.transactions)

        fmt.Println("Previous Hash:", currentBlock.prevHash)

        fmt.Println("Current Hash:", currentBlock.currentHash)

        fmt.Println("-------------------------")

        currentBlock = currentBlock.prevPointer

        blockNumber++

    }

}

// VerifyChain checks if the blockchain is valid

func VerifyChain(chainHead \*Block) {

    // Start from the head of the chain

    currentBlock := chainHead

    // Traverse the blockchain

    for currentBlock != nil {

        // Recalculate the hash of the current block

        calculatedHash := CalculateHash(currentBlock)

        // Check if the calculated hash matches the stored hash

        if calculatedHash != currentBlock.currentHash {

            fmt.Println("Blockchain is compromised! Hash mismatch in block.")

            return

        }

        // Check if the previous block exists

        if currentBlock.prevPointer != nil {

            // Verify that the prevHash of the current block matches the currentHash of the previous block

            if currentBlock.prevHash != currentBlock.prevPointer.currentHash {

                fmt.Println("Blockchain is compromised! Previous hash mismatch.")

                return

            }

        }

        // Move to the previous block

        currentBlock = currentBlock.prevPointer

    }

    // If all blocks are valid

    fmt.Println("Blockchain is unchanged and valid.")

}

func main() {

    // Create the genesis block

    genesisBlock := &Block{

        transactions: []string{"Transaction 1", "Transaction 2"},

        prevPointer:  nil,

        prevHash:     "",

        currentHash:  "",

    }

    genesisBlock.currentHash = CalculateHash(genesisBlock)

    // Insert more blocks

    chainHead := InsertBlock([]string{"Transaction 3", "Transaction 4"}, genesisBlock)

    chainHead = InsertBlock([]string{"Transaction 5", "Transaction 6"}, chainHead)

    // List all blocks

    fmt.Println("Blockchain:")

    ListBlocks(chainHead)

    // Verify the blockchain

    fmt.Println("Verifying blockchain...")

    VerifyChain(chainHead)

    // Change a transaction

    fmt.Println("\nChanging a transaction...")

    ChangeBlock("Transaction 3", "Modified Transaction 3", chainHead)

    // List all blocks again

    fmt.Println("\nBlockchain after modification:")

    ListBlocks(chainHead)

    // Verify the blockchain again

    fmt.Println("Verifying blockchain...")

    VerifyChain(chainHead)

}