GoBlockchain Project Documentation:

This document provides a detailed, technical reference for the GoBlockchain project. It covers the core blockchain logic, the server implementations, and the utility functions that power the entire system.

1. Getting Started

Follow these steps to set up and run the project:

Prerequisites

- **Go:** Version 1.18 or newer.
- **Git:** For cloning the repository.

Setup

1. **Clone the Repository:**
   git clone https://github.com/dkv204p/goblockchain.git
   cd goblockchain

2. **Initialize Modules:**
   go mod init goblockchain
   go mod tidy

3. Run the Servers:
   Open two separate terminals.
   - **Terminal 1 (Blockchain Server):**
     cd blockchain_server
     go run . --port 5000

   - **Terminal 2 (Wallet Server):**

     cd wallet_server
     go run . --port 8080 --gateway http://127.0.0.1:5000

Access the wallet frontend at http://localhost:8080.

2. Core Code Documentation

This section provides a detailed breakdown of every function and struct within the project.

block Package (block/blockchain.go)

This package contains the fundamental logic for the blockchain.

- **consts**: Defines key parameters like MINING_DIFFICULTY, MINING_REWARD, and network configurations.
- **type Block struct**: Represents a single block in the chain.
  - timestamp: The creation time of the block (in nanoseconds since the Unix epoch).
  - nonce: The number used to find the Proof of Work.
  - previousHash: A 32-byte array storing the hash of the preceding block.
  - transactions: A slice of pointers to Transaction structs, representing the data included in the block.
- **func NewBlock(nonce int, previousHash [32]byte, transactions []*Transaction) *Block**:
  - **Description**: A constructor for the Block struct.
  - **Parameters**: nonce, previousHash, transactions.
  - **Returns**: A new *Block instance.
- **func (b *Block) PreviousHash() [32]byte**: A getter for the previousHash field.
- **func (b *Block) Nonce() int**: A getter for the nonce field.
- **func (b *Block) Transactions() []*Transaction**: A getter for the transactions field.
- **func (b *Block) Print()**:
  - **Description**: Prints the contents of a block to the console in a human-readable format.
- **func (b *Block) Hash() [32]byte**:
  - **Description**: Calculates the SHA-256 hash of the block. This hash is used to link blocks together and for Proof of Work validation.
- **func (b *Block) MarshalJSON() ([]byte, error)**:
  - **Description**: Custom JSON marshaller that formats the block's fields, including converting the previousHash byte array to a hexadecimal string.
- **func (b *Block) UnmarshalJSON(data []byte) error**:
  - **Description**: Custom JSON unmarshaller that handles the conversion of the previousHash hexadecimal string back into a byte array.
- **type Blockchain struct**: Represents the entire blockchain data structure and its associated state.
  - transactionPool: A slice of *Transaction structs, holding all pending transactions.
  - chain: A slice of *Block structs, representing the sequence of mined blocks.
  - blockchainAddress: The address of the mining node.

    - port: The port on which the blockchain server is running.
    - mux: A mutex to ensure thread-safe access to the blockchain.
    - neighbors: A slice of strings, holding the addresses of discovered peer nodes.
    - muxNeighbors: A mutex for thread-safe access to the neighbor list.
- **func NewBlockchain(blockchainAddress string, port uint16) *Blockchain**:
    - **Description**: A constructor for the Blockchain struct. It initializes a new blockchain with a genesis block.
- **func (bc *Blockchain) Chain() []*Block**: A getter for the chain field.
- **func (bc *Blockchain) Run()**:
    - **Description**: Starts the background processes for the blockchain, including neighbor synchronization and mining.
- **func (bc *Blockchain) SetNeighbors()**:
    - **Description**: Discovers and sets the list of active peer nodes on the network.
- **func (bc *Blockchain) SyncNeighbors()**:
    - **Description**: A thread-safe method to update the list of neighbors.
- **func (bc *Blockchain) StartSyncNeighbors()**:
    - **Description**: A recursive function that calls SyncNeighbors at a timed interval (BLOCKCHIN_NEIGHBOR_SYNC_TIME_SEC).
- **func (bc *Blockchain) TransactionPool() []*Transaction**: A getter for the transactionPool field.
- **func (bc *Blockchain) ClearTransactionPool()**:
    - **Description**: Resets the transaction pool, typically after a block is mined.
- **func (bc *Blockchain) CreateBlock(nonce int, previousHash [32]byte) *Block**:
    - **Description**: Creates and appends a new block to the blockchain, then clears the transaction pool. It also sends a DELETE request to neighbors to clear their transaction pools.
- **func (bc *Blockchain) LastBlock() *Block**:
    - **Description**: Returns the most recently added block from the chain.
- **func (bc *Blockchain) Print()**:
    - **Description**: Prints the entire blockchain to the console.
- **func (bc *Blockchain) CreateTransaction(sender, recipient string, value float32, senderPublicKey *ecdsa.PublicKey, s *utils.Signature) bool**:
    - **Description**: Adds a new transaction to the pool and propagates it to all neighboring nodes via a PUT request.
- **func (bc *Blockchain) AddTransaction(sender, recipient string, value float32, senderPublicKey *ecdsa.PublicKey, s *utils.Signature) bool**:
    - **Description**: Adds a transaction to the local transaction pool after verifying the signature and the sender's balance.
- **func (bc *Blockchain) VerifyTransactionSignature(senderPublicKey *ecdsa.PublicKey, s *utils.Signature, t *Transaction) bool**:
    - **Description**: Cryptographically verifies the signature of a transaction using the

sender's public key.

- **func (bc *Blockchain) CopyTransactionPool() []*Transaction**:
  - **Description**: Creates a deep copy of the current transaction pool to prevent data race conditions during mining.
- **func (bc *Blockchain) ValidProof(nonce int, previousHash [32]byte, transactions []*Transaction, difficulty int) bool**:
  - **Description**: Checks if a given nonce produces a block hash that meets the mining difficulty criteria (i.e., starts with a certain number of zeros).
- **func (bc *Blockchain) ProofOfWork() int**:
  - **Description**: Finds a valid nonce for the next block by iteratively incrementing it until the ValidProof check passes.
- **func (bc *Blockchain) Mining() bool**:
  - **Description**: The main mining function. It locks the blockchain, adds a reward transaction, performs Proof of Work, creates a new block, and then notifies neighboring nodes of the new block via a PUT request to the /consensus endpoint.
- **func (bc *Blockchain) StartMining()**:
  - **Description**: A recursive function that calls Mining at a timed interval (MINING_TIMER_SEC).
- **func (bc *Blockchain) CalculateTotalAmount(blockchainAddress string) float32**:
  - **Description**: Calculates the balance of a specific blockchain address by iterating through all transactions on the chain.
- **func (bc *Blockchain) ValidChain(chain []*Block) bool**:
  - **Description**: Verifies the integrity of an entire blockchain by checking the hashes and Proof of Work for each block.
- **func (bc *Blockchain) ResolveConflicts() bool**:
  - **Description**: The consensus algorithm. It checks if any neighbor's chain is longer than the current one and, if so, validates and replaces the local chain.
- **type Transaction struct**: Represents a single transaction.
- **type TransactionRequest struct**: A struct used for marshalling and unmarshalling HTTP requests for transactions.
- **type AmountResponse struct**: A struct used for marshalling and unmarshalling HTTP responses for wallet amounts.

blockchain_server Package (blockchain_server/blockchain_server.go)

This package handles the server-side API endpoints for the blockchain.

- **type BlockchainServer struct**: The main server struct.
- **func NewBlockchainServer(port uint16) *BlockchainServer**: Constructor.
- **func (bcs *BlockchainServer) GetBlockchain() *block.Blockchain**:
  - **Description**: Retrieves or initializes the singleton Blockchain instance.
- **func (bcs *BlockchainServer) GetChain(w http.ResponseWriter, req *http.Request)**:
  - **Endpoint**: GET /
  - **Description**: Returns a JSON representation of the entire blockchain.
- **func (bcs *BlockchainServer) Transactions(w http.ResponseWriter, req *http.Request)**:
  - **Endpoint**: GET, POST, PUT, DELETE /transactions
  - **Description**: A multi-method handler for managing the transaction pool.
    - GET: Returns the list of pending transactions.
    - POST: Creates a new transaction and adds it to the pool.
    - PUT: Adds a transaction to the pool (used by other nodes).
    - DELETE: Clears the transaction pool.
- **func (bcs *BlockchainServer) Mine(w http.ResponseWriter, req *http.Request)**:
  - **Endpoint**: GET /mine
  - **Description**: Manually triggers the mining process to create a new block.
- **func (bcs *BlockchainServer) StartMine(w http.ResponseWriter, req *http.Request)**:
  - **Endpoint**: GET /mine/start
  - **Description**: Starts the automatic, timed mining process.
- **func (bcs *BlockchainServer) Amount(w http.ResponseWriter, req *http.Request)**:
  - **Endpoint**: GET /amount
  - **Description**: Calculates and returns the balance of a given blockchain address from a query parameter.
- **func (bcs *BlockchainServer) Consensus(w http.ResponseWriter, req *http.Request)**:
  - **Endpoint**: PUT /consensus
  - **Description**: Triggers the consensus algorithm to synchronize the local chain with the longest one on the network.
- **func (bcs *BlockchainServer) Run()**:
  - **Description**: Starts the HTTP server and registers all the endpoint handlers.

wallet Package (wallet/wallet.go)

This package contains the logic for creating and managing wallets and transactions.

- **type Wallet struct**: Represents a wallet with private/public key pairs and a blockchain address.
- **func NewWallet() *Wallet**:
  - **Description**: Generates a new wallet using ECDSA and a series of hashing algorithms (SHA-256 and RIPEMD-160) to create a Base58Check-encoded address.
- **func (w *Wallet) PrivateKey() *ecdsa.PrivateKey**: Getter.
- **func (w *Wallet) PrivateKeyStr() string**: Getter, returns the private key as a hexadecimal string.
- **func (w *Wallet) PublicKey() *ecdsa.PublicKey**: Getter.
- **func (w *Wallet) PublicKeyStr() string**: Getter, returns the public key as a hexadecimal string.
- **func (w *Wallet) BlockchainAddress() string**: Getter, returns the Base58-encoded address.
- **func (w *Wallet) MarshalJSON() ([]byte, error)**: Custom JSON marshaller for the Wallet struct.
- **type Transaction struct**: A struct that wraps the transaction details and the sender's keys for signing.
- **func NewTransaction(...) *Transaction**: Constructor for Transaction.
- **func (t *Transaction) GenerateSignature() *utils.Signature**:
  - **Description**: Uses the sender's private key to cryptographically sign a transaction.
- **type TransactionRequest struct**: A struct for unmarshalling transaction data from the wallet frontend.

wallet_server Package (wallet_server/wallet_server.go)

This package runs a local web server to provide a user interface for wallet interactions.

- **type WalletServer struct**: The main server struct.
- **func NewWalletServer(port uint16, gateway string) *WalletServer**: Constructor.
- **func (ws *WalletServer) Index(w http.ResponseWriter, req *http.Request)**:
  - **Endpoint**: GET /
  - **Description**: Serves the index.html file to the browser.
- **func (ws *WalletServer) Wallet(w http.ResponseWriter, req *http.Request)**:
  - **Endpoint**: POST /wallet
  - **Description**: Generates a new wallet and returns its details as JSON.
- **func (ws *WalletServer) CreateTransaction(w http.ResponseWriter, req *http.Request)**:
  - **Endpoint**: POST /transaction
  - **Description**: Receives a transaction request, signs it with the private key, and forwards it to the blockchain server's /transactions endpoint.
- **func (ws *WalletServer) WalletAmount(w http.ResponseWriter, req *http.Request)**:
  - **Endpoint**: GET /wallet/amount
  - **Description**: Queries the blockchain server's /amount endpoint to get and return a wallet's current balance.
- **func (ws *WalletServer) Run()**:
  - **Description**: Starts the HTTP server for the wallet and registers all the handlers.

utils Package

Contains various utility functions used throughout the project.

- **utils/ecdsa.go**:
  - **type Signature struct**: Holds the R and S components of an ECDSA signature.
  - **func (s *Signature) String() string**: Formats the signature as a single hexadecimal string.
  - **func String2BigIntTuple(s string) (big.Int, big.Int)**: Parses a hexadecimal string back into two big.Int values.
  - **func SignatureFromString(s string) *Signature**: Converts a signature string to a Signature struct.
  - **func PublicKeyFromString(s string) *ecdsa.PublicKey**: Converts a public key string to an ecdsa.PublicKey struct.
  - **func PrivateKeyFromString(s string, publicKey *ecdsa.PublicKey) *ecdsa.PrivateKey**: Converts a private key string to an ecdsa.PrivateKey struct.
- **utils/json.go**:
  - **func JsonStatus(message string) []byte**: Creates a simple JSON status response.
- **utils/neighbor.go**:
  - **func IsFoundHost(host string, port uint16) bool**: Checks if a given host and port are reachable.
  - **func FindNeighbors(...) []string**: Discovers potential peer nodes by scanning a range of IP addresses and ports.
  - **func GetHost() string**: Retrieves the local machine's IP address.

3. API Reference Summary

This section provides a quick, structured overview of all available endpoints.

Service	HTTP Method	Endpoint	Description
Blockchain	GET	/chain	Get the full blockchain.
Blockchain	GET	/transactions	Get the list of pending transactions.
Blockchain	POST	/transactions	Create a new transaction.
Blockchain	PUT	/consensus	Synchronize the blockchain with peers.
Blockchain	GET	/mine	Manually mine a new block.
Blockchain	GET	/mine/start	Start automatic mining.
Blockchain	GET	/amount	Get the balance of a wallet.
Wallet	GET	/	Serve the wallet UI.
Wallet	POST	/wallet	Generate a new wallet.
Wallet	POST	/transaction	Create and sign a transaction.
Wallet	GET	/wallet/amount	Get the wallet's balance.

4. Conclusion

This project is a robust, educational example of a blockchain implementation. By carefully examining each function and its role, you can gain a deep understanding of core blockchain principles such as Proof of Work, cryptography, and network consensus. The modular design, with separate servers for the blockchain and wallet, provides a clear separation of concerns.