Problem Statements :

As a group of business partners, we want to be able to securely manage our shared funds and make sure that multiple people need to approve any transactions, so that no single person can make decisions on their own.

As an individual who is part of a non-profit organization, I want to be able to contribute funds to the organization's account and be able to see all the transactions that are made from that account, so that I can have transparency and accountability.

As a group of friends who are investing in a new venture, we want to be able to pool our funds together and have a shared wallet that requires multiple confirmations for any transactions, so that we can ensure that no one person can make decisions on their own and that our funds are secure.

As a company that wants to pay our employees with cryptocurrency, we want to be able to manage our funds securely and ensure that only authorized personnel can make transactions, so that we can protect our assets and maintain control over our finances.

As an individual who is part of a trading group, I want to be able to manage our shared funds in a secure way and require multiple confirmations for any transactions, so that we can prevent any fraud or misuse of our funds.

This Solution can benefit and used by business partners in a firm , member of non profit organization , group of friends who are investing in a new venture to mange funds, individual who want to ensure and manage our shared funds in a secure way

Architecture Components

User Interface: This is the front-end component of the wallet where users interact with the system. It includes the web or mobile app that allows users to view their account balances, send or receive funds, and approve or reject transactions.

Backend Server: This is the core component of the wallet that handles all the business logic, data storage, and processing of user transactions. It consists of a server that runs the blockchain nodes and the smart contracts responsible for implementing the multisignature wallet functionality.

Smart Contract: This is the code that runs on the blockchain network and defines the rules and logic for the multisignature wallet. It implements the features such as adding and removing wallet owners, submitting transactions, and approving transactions.

Blockchain Network: This is the decentralized, distributed network that powers the multisignature wallet. It is responsible for recording all transactions and ensuring that the system is secure and tamper-proof.

External Services: Depending on the specific implementation, multisignature wallets may require integration with external services such as KYC/AML verification services, payment gateways, or identity verification services.

Problem statement Breakdown in a very High level Business Functional Requirement:

Multiple sender --> Single Reciever based on the confirmations from different sender participants

Allows multiple owners to jointly control the wallet and require a certain number of confirmations before executing a transaction.

To achieve this implemented a multi-signature wallet that requires multiple owners to confirm transactions before they can be executed and
contratct stores a list of owners and requires a specified number of confirmations from owners before a transaction can be executed.

Technologies used : Solidity,Remix,Node.js,Testnet

Solidity Concepts Used in this project : array, mapping, modifier, events, structs, mapping, payable functions, require

Arrays and Mappings: Arrays and mappings are used to store and retrieve data in the smart contract. The owners array is used to store the addresses of the contract owners, while the isOwner mapping is used to check if an address is a contract owner.

Modifiers: Modifiers are used to restrict access to certain functions in the smart contract based on certain conditions. For example, the onlyOwner modifier restricts access to certain functions to only the contract owners. The modifier keyword is used to define modifiers in Solidity.

Structs: Structs are used to define custom data types in Solidity. In the above smart contract, a Transaction struct is defined to represent a transaction that can be submitted to the multi-signature wallet.

Events: Events are used to emit notifications when certain actions occur in the smart contract, such as when a transaction is submitted, confirmed, revoked, or executed. The event keyword is used to define events in Solidity.

Payable functions: Payable functions are used to receive Ether in the smart contract. In the above smart contract, the DepositETH() and receive() functions are defined as payable functions to receive Ether.

require statement used for validation and to check if certain conditions are met before executing the function further. If the condition specified in the require statement evaluates to false, then the function will immediately stop execution and any changes made so far in the function will be reverted.

Breakdown of Functional Requiremens to Features to implement:

Create the contract with necessary variables, data structures and mappings.
Implement constructor to initialize the contract with the required number of confirmations and owners.

Implement submitTransaction() function to allow owners to submit transactions to the wallet.
Implement confirmTransaction() function to allow owners to confirm a transaction.
Implement executeTransaction() function to execute a transaction once the required number of confirmations are received.
Implement revokeConfirmation() function to allow owners to revoke their confirmation for a transaction.

Implement getOwners() function to allow anyone to retrieve the list of owners.
Implement getTransactionCount() function to allow anyone to retrieve the number of transactions submitted.
Implement getTransaction() function to allow anyone to retrieve the details of a particular transaction.
Implement DepositETH() function to allow anyone to fund the contract.

The number of owners required to confirm a transaction is specified in the constructor and stored in the numConfirmationsRequired variable. The value is passed as an argument when the contract is deployed and must be greater than 0 and less than or equal to the total number of owners.

7 functions in the contract and this contract emits several events.

Deposit: emitted when someone deposits ether into the contract
SubmitTransaction: emitted when an owner submits a new transaction
ConfirmTransaction: emitted when an owner confirms a transaction
RevokeTransaction: emitted when an owner revokes their confirmation of a transaction
ExecuteTransaction: emitted when a transaction is executed

Value it creates in Adaption :

The adoption value of multisign wallets lies in their ability to provide enhanced security and trust to users. With multisign wallets, users can have confidence that their transactions will not be carried out without their explicit authorization, or that of other authorized parties.

This feature is particularly valuable for businesses or organizations that handle large amounts of funds, as it provides an added layer of protection against potential theft or fraud.

Furthermore, multisign wallets can also facilitate more complex transactions, such as those involving escrow or trust arrangements. By requiring multiple signatures or approvals, these wallets can help ensure that all parties involved in a transaction are in agreement and that the transaction is carried out fairly and transparently.

Overall, the adoption of multisign wallets can help to increase the security and reliability of high value transactions, and provide users with greater confidence in the safety and integrity of their assets.