Vivekanand Education Society's Institute of Technology, Chembur, Mumbai,
Department of Computer Engineering
Year : 2024-25 (ODD Sem)
MID TERM TEST

Class : BE	**Division: All Branches (Honor/ Minor Degree)**
Semester: VII	**Subject: Blockchain Development**
Date: 6th Sept 2024	**Time: 1pm to 2pm**

Q1 a. With an example explain function modifiers in Solidity.

Function modifiers in Solidity are a way to change the behavior of functions in a smart contract. They are used to enforce rules, validate conditions, or modify the behavior of functions. Modifiers are defined using the modifier keyword and can be applied to functions to alter their execution.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract AccessControl {
    address public owner;

    constructor() {
        owner = msg.sender; // Set the contract deployer as the owner
    }
    // Modifier to restrict access to the owner
    modifier onlyOwner() {
        require(msg.sender == owner, "You are not the owner");
        _; // This is a placeholder for the function body where the modifier is applied
    }
    // Function to change the owner
    function changeOwner(address newOwner) public onlyOwner {
        owner = newOwner;
    }
    // Function that any user can call
    function publicFunction() public pure returns (string memory) {
        return "This is a public function";
    }
}
```

When changeOwner is called, the onlyOwner modifier first executes its code. If the condition msg.sender == owner is satisfied, the _ placeholder is replaced with the body of the changeOwner function. If the condition fails, the function call is reverted, and the changeOwner function does not execute. This pattern helps in ensuring that only authorized users can perform certain actions, which is crucial for maintaining security in smart contracts.

Q1 b. Explain the role of IPFS in the Blockchain Environment

IPFS, which stands for InterPlanetary File System, plays a complementary role to blockchain technology, enhancing its capabilities in several key ways.

1. Decentralized Storage: IPFS is a peer-to-peer network designed for decentralized file storage and sharing. Unlike traditional file systems that rely on central servers, IPFS distributes files across a network of nodes. Each file is broken into smaller chunks, hashed, and stored across various nodes. This ensures that files are not stored in a single location but are spread across the network, improving resilience and availability.

2. Efficient Data Retrieval: In a blockchain environment, especially when dealing with large amounts of data or complex documents, IPFS can store and manage these data off-chain. The blockchain itself often has limitations in terms of storage space and performance. IPFS addresses this by allowing files to be retrieved quickly and efficiently through its content-addressing system, where each file is identified by a unique cryptographic hash.

3. Content Addressing: IPFS uses content-based addressing instead of location-based addressing (like traditional URLs). This means that files are referenced by their unique hash, ensuring that the data retrieved is exactly the data that was intended. This hash-based addressing is particularly useful in blockchain applications where data integrity and immutability are crucial.

4. Integration with Smart Contracts: In blockchain environments that use smart contracts, such as Ethereum, IPFS can be used to store data off-chain while referencing it on-chain. Smart contracts can store the IPFS hash of the data, ensuring that the contract can access or verify the data without storing it directly on the blockchain. This helps in reducing the load and cost of storing large data on-chain.

5. Censorship Resistance: IPFS contributes to censorship resistance by distributing data across many nodes. This decentralized approach makes it more difficult for any single entity to censor or control the availability of the data, complementing the censorship resistance offered by blockchain technology.

6. Data Integrity and Verification: Since IPFS relies on cryptographic hashes, the integrity of data can be verified easily. If someone tries to alter a file, its hash will change, and the new data will have a different hash. This feature aligns well with the principles of blockchain, where ensuring data integrity is fundamental.

7. Scalability: Blockchain systems can become congested and expensive when dealing with large volumes of data. IPFS can offload much of this data storage and retrieval, improving the overall scalability of blockchain applications. By handling large files and data separately, IPFS allows blockchains to focus on transaction processing and consensus.

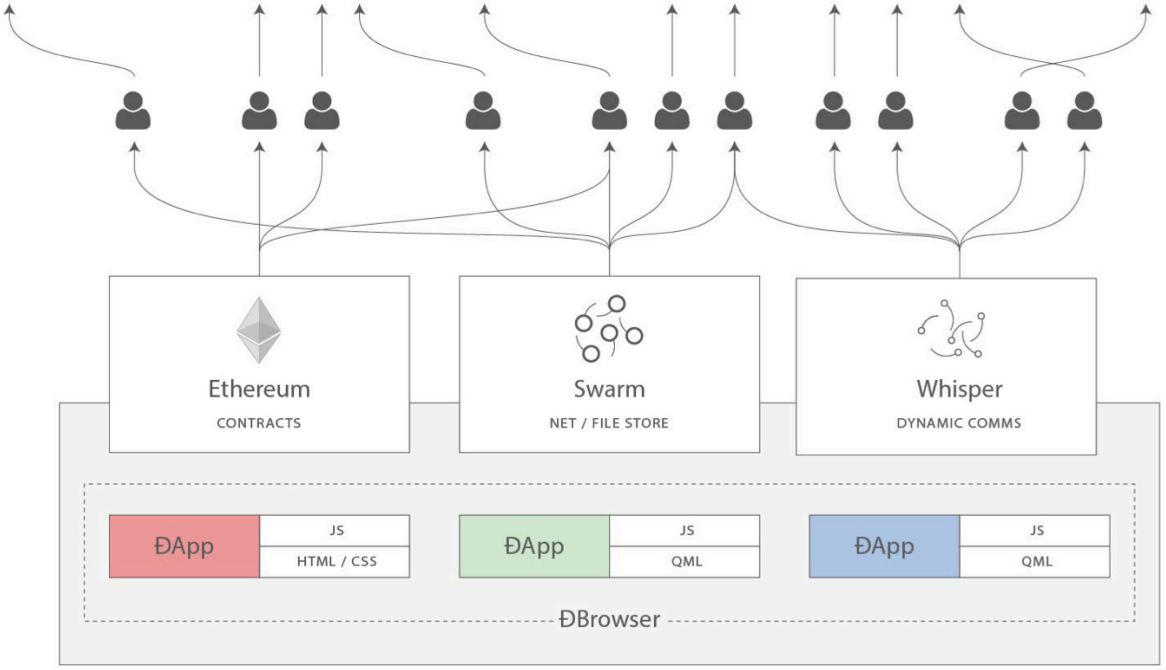
Vivekanand Education Society's Institute of Technology, Chembur, Mumbai,
Department of Computer Engineering
Year : 2024-25 (ODD Sem)
MID TERM TEST

Q1 c. Compare Blockchain programming languages - Solidity and Go

Parameter	Solidity	Go (Golang)
Purpose	Primarily used for writing smart contracts on Ethereum blockchain and other EVM-based platforms.	General-purpose programming language used in a variety of applications, including blockchain (e.g., Hyperledger Fabric).
Syntax	Similar to JavaScript, with a statically-typed structure.	C-like syntax, statically-typed, and focuses on simplicity and readability.
Blockchain Platforms	Ethereum, Binance Smart Chain, Polygon, and other EVM-compatible platforms.	Hyperledger Fabric, Cosmos, and other blockchain frameworks that support Go.
Compilation	Compiled to bytecode for the Ethereum Virtual Machine (EVM).	Compiled to native binaries. Can also compile to WebAssembly (Wasm) for use in blockchain environments.
Execution Environment	Runs on the Ethereum Virtual Machine (EVM).	Runs on native environments or within containerized environments in blockchain platforms like Hyperledger Fabric.
Gas Optimization	Requires careful gas optimization to minimize transaction costs on Ethereum.	Not specifically tied to gas costs, but efficiency is still important in blockchain contexts.
Concurrency Support	No built-in concurrency; relies on EVM's single-threaded execution model.	Strong concurrency support with Goroutines, making it highly suitable for concurrent blockchain operations.
Security Considerations	High; specific to smart contract vulnerabilities like reentrancy, integer overflows, etc.	General programming security, along with blockchain-specific concerns like chaincode security.
Learning Curve	Moderate to steep for those unfamiliar with JavaScript or blockchain concepts.	Moderate; easier for developers familiar with C-like languages, but requires understanding of concurrency and blockchain concepts.
Community and Ecosystem	Large and active community focused on Ethereum development.	Large and diverse community with extensive resources for general Go development, with growing support in blockchain.

Testing Frameworks	Truffle, Hardhat, Brownie.	Go's built-in testing framework, with additional tools like Ginkgo, Gomega for blockchain testing.
Development Tools	Remix, Truffle, Hardhat, OpenZeppelin.	Visual Studio Code, GoLand, Docker for containerization, Hyperledger Fabric SDK.
Interoperability	Primarily interoperable within EVM-based chains.	Highly interoperable across various platforms, including non-blockchain applications.
Deployment	Deployed on Ethereum or other EVM-compatible blockchains via specific deployment scripts.	Deployed in Hyperledger Fabric as chaincode or in Cosmos SDK-based blockchains.
Contract Upgradability	Complex; requires proxy contracts or other patterns for upgradeability.	Flexible; chaincode can be upgraded more straightforwardly in platforms like Hyperledger Fabric.

Q1d. With neat diagram explain whisper and swarm



Whisper is a communication protocol designed for private, secure, and decentralized messaging on the Ethereum network. It allows DApps (Decentralized Applications) to send messages to each other securely without relying on centralized servers. Whisper is part of the Ethereum stack and is used to enable real-time communication between nodes in a decentralized manner.

Swarm is a decentralized storage platform and content distribution service. It is intended to provide a decentralized way to store and distribute large amounts of data, such as DApp files, off-chain data, and other resources that are too large to store on the Ethereum blockchain. Swarm works by breaking up files into smaller chunks, distributing them across multiple nodes, and ensuring that the data remains accessible and resilient.

Q1e. Explain the different types of Ethereum Clients

Ethereum clients are software implementations that allow nodes to interact with the Ethereum network, enabling them to validate transactions, execute smart contracts, and maintain the blockchain's state. There are several types of Ethereum clients, each serving different purposes and tailored for different use cases.

1. Full Nodes (Full Clients) : store and validate the entire blockchain history from the genesis block to the latest block. They ensure the network's security and decentralization by independently verifying each transaction and block.

- Key Features:
    - Complete Blockchain History: Stores all past transactions and smart contract states.
    - Verification: Independently verifies the validity of all blocks and transactions.
    - Network Security: Contributes to the network's security by participating in consensus.
- Examples:
    - Geth (Go-Ethereum): The most popular Ethereum client, written in Go.
    - Nethermind: Written in .NET, known for its performance and modularity.
    - Besu: An enterprise-grade Ethereum client written in Java, part of the Hyperledger project.
    - Erigon: A more efficient re-implementation of Geth, focusing on performance and storage optimization.

2. Light Nodes (Light Clients) : only store the headers of blocks and rely on full nodes to provide the necessary data to verify transactions. This type of client is suitable for devices with limited storage or computational power.

- Key Features:
    - Minimal Storage: Stores only block headers, not the full blockchain.
    - Dependency on Full Nodes: Requests data from full nodes to verify transactions.
    - Lower Resource Requirements: Ideal for mobile devices and lightweight applications.
- Examples:
    - Geth Light Mode: Geth can operate in a light mode, serving as a light client.
    - Prysm Light Client: Part of the Prysm Ethereum 2.0 client, used for Ethereum's proof-of-stake network.

3. Archive Nodes : store everything that a full node stores, plus they retain historical state data. This includes the state of every account at every block height. Archive nodes are often used for specialized purposes like data analysis, historical queries, and blockchain explorers.

- Key Features:
    - Complete Historical State: Stores the full history of the blockchain, including all states at every block.

    - Data Retrieval: Used for applications requiring detailed historical data, such as blockchain explorers.
    - High Storage Requirement: Requires significant disk space due to the massive amount of data stored.
- Examples:
    - Geth Archive Mode: Geth can be run in archive mode, storing all historical states.
    - Parity (OpenEthereum) Archive: Parity can also be run in archive mode for historical data.

Q1 f. Enlist five opportunities and challenges associated with Smart Contracts

Opportunities of Smart Contracts:

1. Automation: Smart contracts enable the automation of processes, reducing the need for intermediaries and human intervention.
2. Cost Efficiency: By eliminating middlemen, smart contracts can lower transaction and operational costs.
3. Transparency: Smart contracts are stored on a blockchain, ensuring transparency as all participants can view the contract terms.
4. Security: Being built on blockchain technology, smart contracts benefit from the security and immutability of the ledger.
5. Speed: Transactions and agreements can be executed instantly once conditions are met, speeding up processes.

Challenges of Smart Contracts:

1. Legal Uncertainty: There is often ambiguity around the legal status and enforceability of smart contracts across different jurisdictions.
2. Coding Flaws: Errors in code can lead to vulnerabilities, exposing contracts to bugs or hacks.
3. Scalability Issues: Processing a large number of smart contracts simultaneously may lead to performance bottlenecks in blockchain networks.
4. Limited Flexibility: Once deployed, smart contracts are difficult to modify, which can be a challenge if terms need adjustments.
5. Interoperability: Ensuring smart contracts can interact with different blockchain platforms and systems remains a technical challenge.

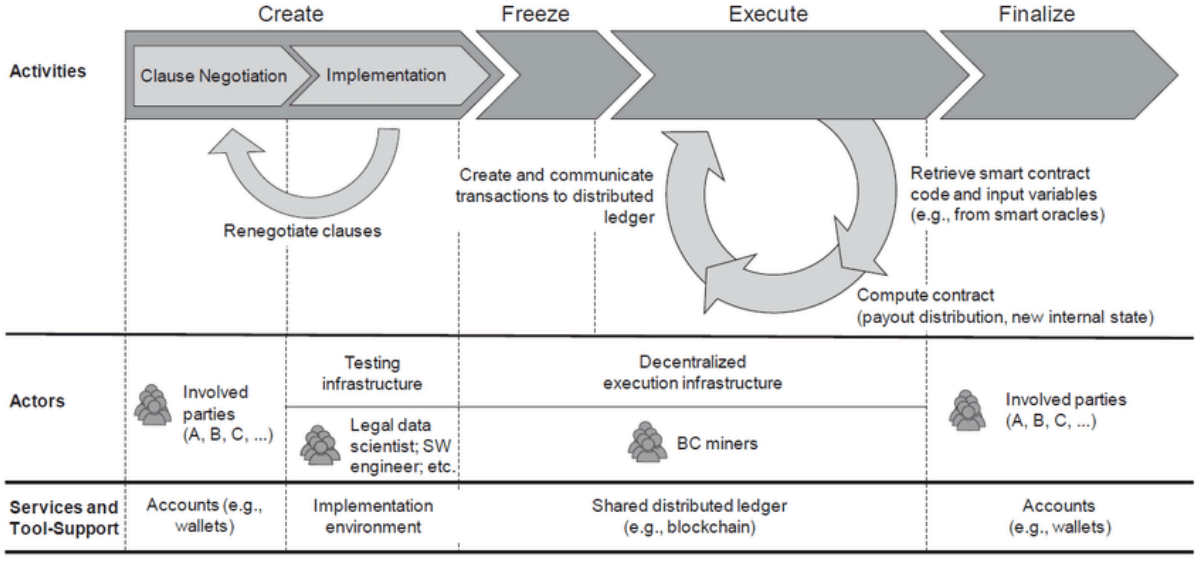
Q2 a. Explain the ERC20 Token Functions

ERC20 is a popular standard for creating tokens on the Ethereum blockchain. It defines a set of functions that allow for standardization across all ERC20 tokens, ensuring interoperability with wallets, exchanges, and other smart contracts. Below are the key ERC20 functions:

1. totalSupply()
    - Description: Returns the total number of tokens that exist.
    - Purpose: Helps users and smart contracts understand the overall supply of tokens in circulation.
2. balanceOf(address owner)
    - Description: Returns the balance of tokens for a given address.
    - Purpose: Allows users and smart contracts to check how many tokens are held by a specific account.

3. transfer(address to, uint256 amount)
    - Description: Transfers a certain amount of tokens from the sender to another address.
    - Purpose: Enables token holders to send tokens directly to another Ethereum address.
4. approve(address spender, uint256 amount)
    - Description: Allows an address (spender) to withdraw tokens from the owner's account, up to the specified amount.
    - Purpose: Useful in situations where a smart contract or third party needs to spend tokens on behalf of the token holder.
5. transferFrom(address from, address to, uint256 amount)
    - Description: Allows an approved spender to transfer tokens from one address to another.
    - Purpose: This is used in conjunction with approve() to enable authorized transfers, often in decentralized exchanges or payment gateways.
6. allowance(address owner, address spender)
    - Description: Returns the number of tokens that a spender is still allowed to withdraw from the owner's account.
    - Purpose: Helps check the remaining tokens that an approved spender can use from an owner's account under the approve() function.

Q2 b. Using a neat diagram explain the lifecycle of a Smart Contract



1. Creation Phase:
    a. consists of iterative contract negotiation and an implementation phase.
    b. First, the parties must agree on the broad content and goals of the contract.
    c. Similar to typical contract negotiations and can be conducted online or in person contracts
    d. During this phase, the following tasks are completed:

i. Multiple-party bargaining.
ii. Design, implementation, and validation of smart

2. Freeze:
    a. The validation of transactions on a blockchain is performed by a network of computers known as nodes. The blockchain miners are these nodes.
    b. To prevent the ecosystem from being swamped with smart contracts, miners must be paid a tiny fee in exchange for this service.
3. Execution:
    a. Contracts placed on the distributed ledger are read by participating nodes.
    b. The authentication nodes validate the integrity of a smart contract, the code is performed by the smart contract interference engine (or by the compiler).
    c. When one party's inputs for execution are received in the form of coins (commitment to goods via coins), the interference engine generates a transaction triggered by the met criterion.
    d. The execution of the smart contract results in a new set of transactions and a new state for the smart contract.
    e. The discoveries and new state information are entered into the distributed ledger and validated using the consensus procedure.
4. Finalize:
    a. The resulting transactions and updated state information are recorded in the distributed ledger and confirmed via the consensus process.
    b. The previously pledged digital assets are transferred (assets are unfrozen), and the contract is signed to confirm all transactions.

Q3 a. Write a smart contract to demonstrate address data type in Solidity

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract AddressDemo {
  // State variable to store an Ethereum address
  address public owner;
  address public savedAddress;
  // Constructor to set the contract deployer's address as the owner
  constructor() {
    owner = msg.sender;  // msg.sender is the address that deploys the contract
  }
  // Function to store an address
  function setAddress(address _address) public {
    savedAddress = _address;
  }
  // Function to get the saved address
  function getAddress() public view returns (address) {
    return savedAddress;
  }
  // Function to check if the caller is the contract owner
```

```
    function isOwner() public view returns (bool) {
        return msg.sender == owner;
    }
    // Function to return the balance of a specific address
    function getBalance(address _address) public view returns (uint256) {
        return _address.balance;
    }
}
```

State Variables:

- owner: Stores the address of the contract owner (the one who deployed the contract).
- savedAddress: Holds an Ethereum address provided by the user.

Functions:

- setAddress(address _address): Saves the input address to the state variable savedAddress.
- getAddress(): Returns the saved address.
- isOwner(): Checks whether the caller (using msg.sender) is the contract owner and returns true if so.
- getBalance(address _address): Returns the Ether balance of any provided Ethereum address.

Q3 b. With the help of a neat diagram explain the role of smart contracts in Supply chain Management systems

Smart contracts play a crucial role in improving the efficiency, transparency, and reliability of Supply Chain Management (SCM) systems. By automating various processes and ensuring trust through decentralized execution, they help streamline operations, reduce fraud, and ensure data integrity. Role of Smart Contracts in Supply Chain Management Systems:

1. **Supplier Contract Automation:**
    - Smart contracts can automatically execute agreements between suppliers and buyers when predefined conditions are met (e.g., delivery of goods, quality control, etc.).
2. **Tracking and Traceability:**
    - Smart contracts record the movement of goods at every stage, from production to delivery. This ensures full traceability of items across the supply chain.
3. **Payment Automation:**
    - Payments to suppliers or logistics providers can be automatically triggered upon delivery verification, reducing delays and ensuring timely payments.
4. **Quality Assurance:**
    - Smart contracts can validate whether goods meet specified quality standards before moving to the next stage in the supply chain, preventing faulty products from advancing.
5. **Inventory Management:**
    - Automated reordering processes can be initiated via smart contracts when inventory levels fall below a specific threshold.
6. **Dispute Resolution:**

- Smart contracts provide immutable proof of transactions, minimizing disputes regarding delivery, payments, and contract fulfillment.

The diagram would visually represent a supply chain flow, where each step—such as supplier agreements, manufacturing, warehousing, and delivery—is connected by smart contracts. These smart contracts automate tasks like quality control, shipment tracking, and payment upon receipt, ensuring transparency and trust throughout the supply chain.

++++++++++++++++++++++**Happy Learning!!!**++++++++++++++++++++++