BCM Syllabus for 2nd IA

Unit – II	08 Hours
Smart contracts: The concept of a smart contract; Design of a smart contract; Development of smart contract code; Deploying and testing the smart contract; Decentralized airline system use case; Airlines smart contract; Motivating decentralized scenarios; Smart contract design considerations; Best practices	

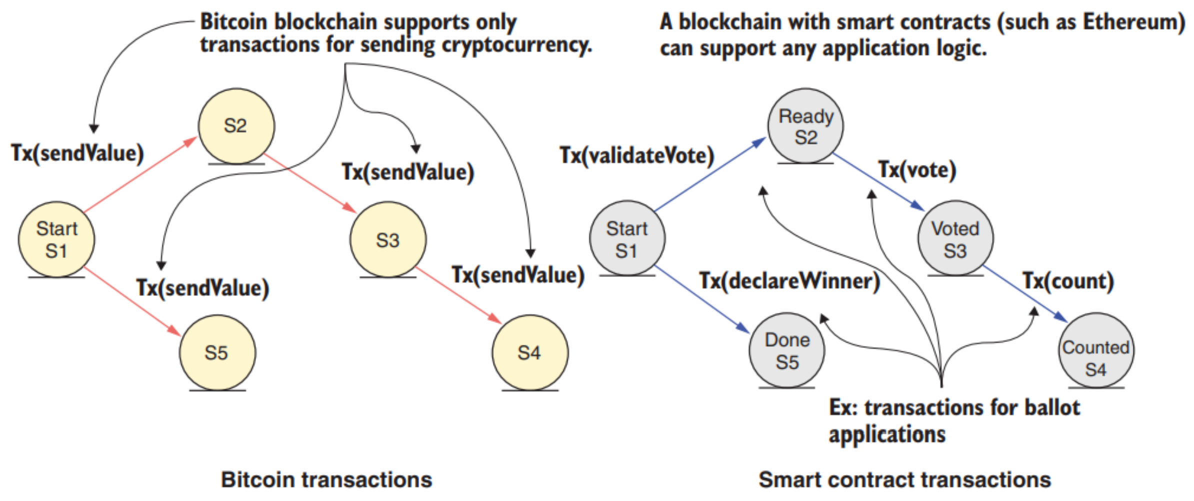
Unit – III	08 Hours
Techniques for trust and integrity: Essentials of trust and integrity; Implementing trust intermediation; Testing; Establishing trust with modifiers, require(), revert(), and assert(); Best practices From smart contracts to Dapps: Preliminary concepts; Dapp development using the Truffle IDE; Installing the Ganache test chain; Smart contract development; Dapp web application development; Introspection; Best practices	

Unit – IV	08 Hours
Security and privacy: Deploying smart contracts on Ropsten; Cryptography basics; Application of public key cryptography	

BCM Question Bank

1. **Explain with diagram Cryptocurrency transactions versus smart contract transactions.**

Let's compare a Bitcoin transaction and a smart contract transaction, as shown in figure 2.2, to give you an idea of the difference between currency transactions and noncurrency, application-dependent function calls. As you can see, in Bitcoin, all the transactions are about sending value (Tx(sendValue)). In the case of a blockchain that supports smart contracts, a transaction embeds a function implemented by the smart contract. In figure 2.2, this function is a voting smart contract. The functions are validateVoter(), vote(), count(), and declareWinner(). The invocation of these functions results in the transactions that will be recorded on the blockchain (Tx(validateVoter), Tx(vote), and so on). This ability to deploy an arbitrary logic on a blockchain significantly enhances its applicability beyond simple cryptocurrency transfers.


Bitcoin blockchain supports only transactions for sending cryptocurrency.
A blockchain with smart contracts (such as Ethereum) can support any application logic.
Tx(sendValue)
Start S1
S2
S3
S4
S5
Tx(sendValue)
Tx(sendValue)
Tx(sendValue)
Tx(validateVote)
Ready S2
Tx(vote)
Start S1
Voted S3
Tx(declareWinner)
Tx(count)
Done S5
Counted S4
Ex: transactions for ballot applications
Bitcoin transactions
Smart contract transactions

2. **Explain with diagram Blockchain application stack and layers**

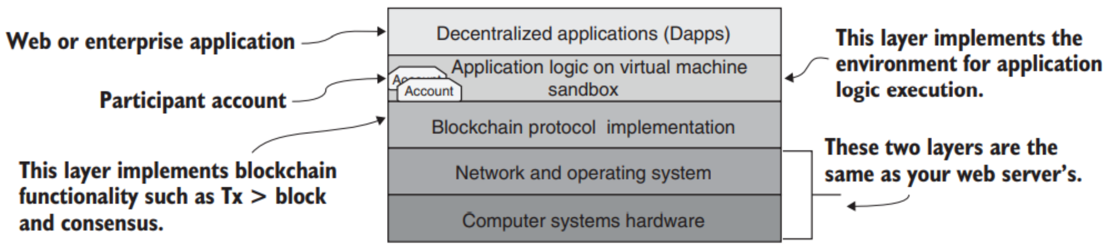

Web or enterprise application
Participant account
This layer implements blockchain functionality such as Tx > block and consensus.
Decentralized applications (Dapps)
Application logic on virtual machine sandbox
Account
Blockchain protocol implementation
Network and operating system
Computer systems hardware
This layer implements the environment for application logic execution.
These two layers are the same as your web server's.

Figure 2.1 Blockchain application stack and layers

Let's start from the bottom and move up. The lower two levels are the standard hard ware and software of most computing systems. The next level up is the blockchain pro tocol level: it houses the components of the blockchain, but you won't program at this level. The next layer hosts the application logic. This layer is where you solve problems like access control to data and code functions for validation, verification, and record ing. The top layer is the user-facing interface where web (or enterprise) programming is done, such as with HTML, JavaScript, and associated frameworks. These elements form the Dapp and its user interface (UI) layer.

3. **What are the different functions for Use case diagram for the counter.**

The UML use case diagram helps you think through the problem and decide how the smart contract—and, more specifically, its functions—will be used. Figure 2.3 shows just one actor: a stick figure representing a decentralized application that will use the counter.

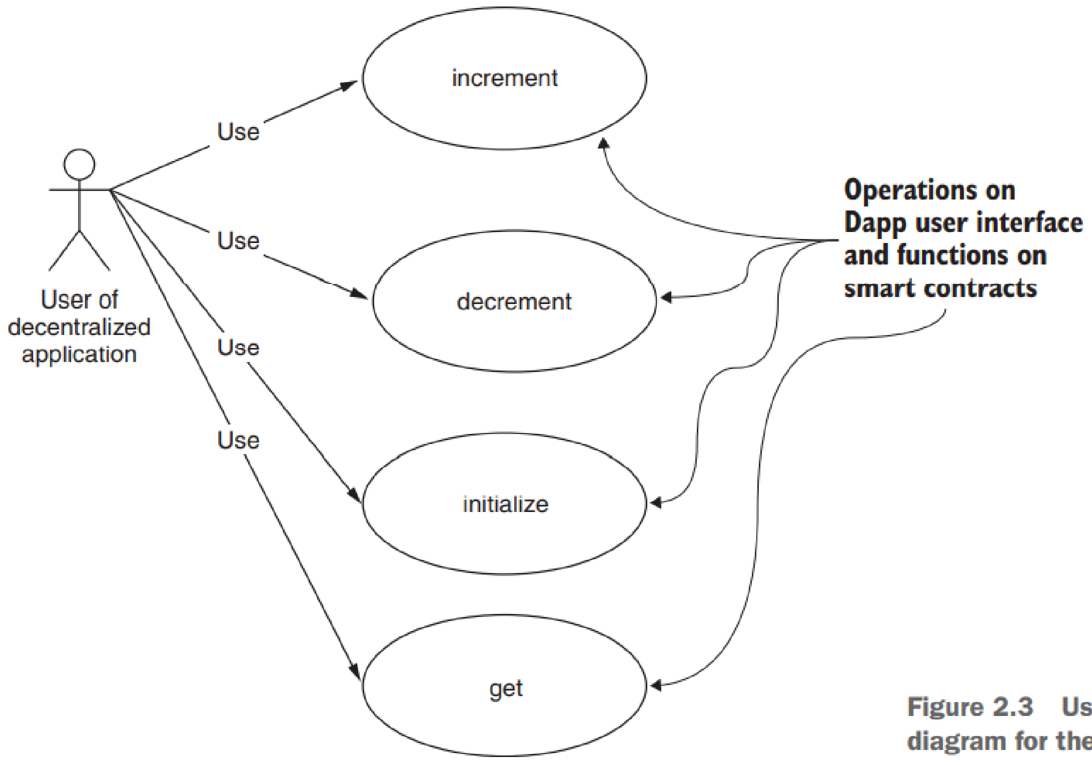

User of decentralized application
Use
Use
Use
Use
increment
decrement
initialize
get
Operations on Dapp user interface and functions on smart contracts

Figure 2.3 Use case diagram for the counter

First, let's think about the functions of a counter:
=> initialize() to a value.
=> increment() by a value.
=> decrement() by a value.
=> get() to access the value of the counter.

This diagram clearly articulates the intent of the smart contract. This diagram is a good starting point for the design process, providing an artifact for discussion among your team members and stakeholders who are interested in the problem. It also pro vides a systematic lead-in to the next steps in the design process. Note that this step of use case design representation is dependent on the problem specification; it does not require you to specify any coding or system dependencies.

4. **Compare Class diagram versus contract diagram templates.**

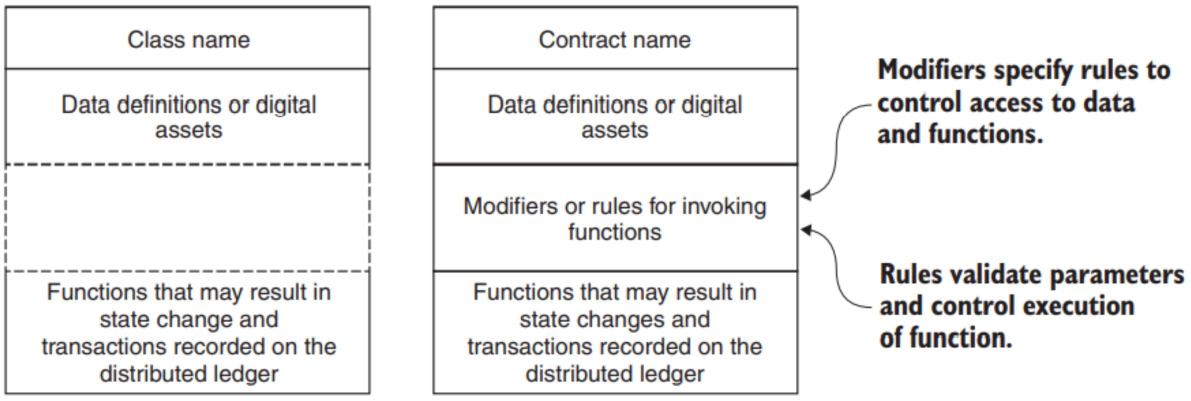

Class name
Data definitions or digital assets
Functions that may result in state change and transactions recorded on the distributed ledger
Contract name
Data definitions or digital assets
Modifiers or rules for invoking functions
Functions that may result in state changes and transactions recorded on the distributed ledger
Modifiers specify rules to control access to data and functions.
Rules validate parameters and control execution of function.

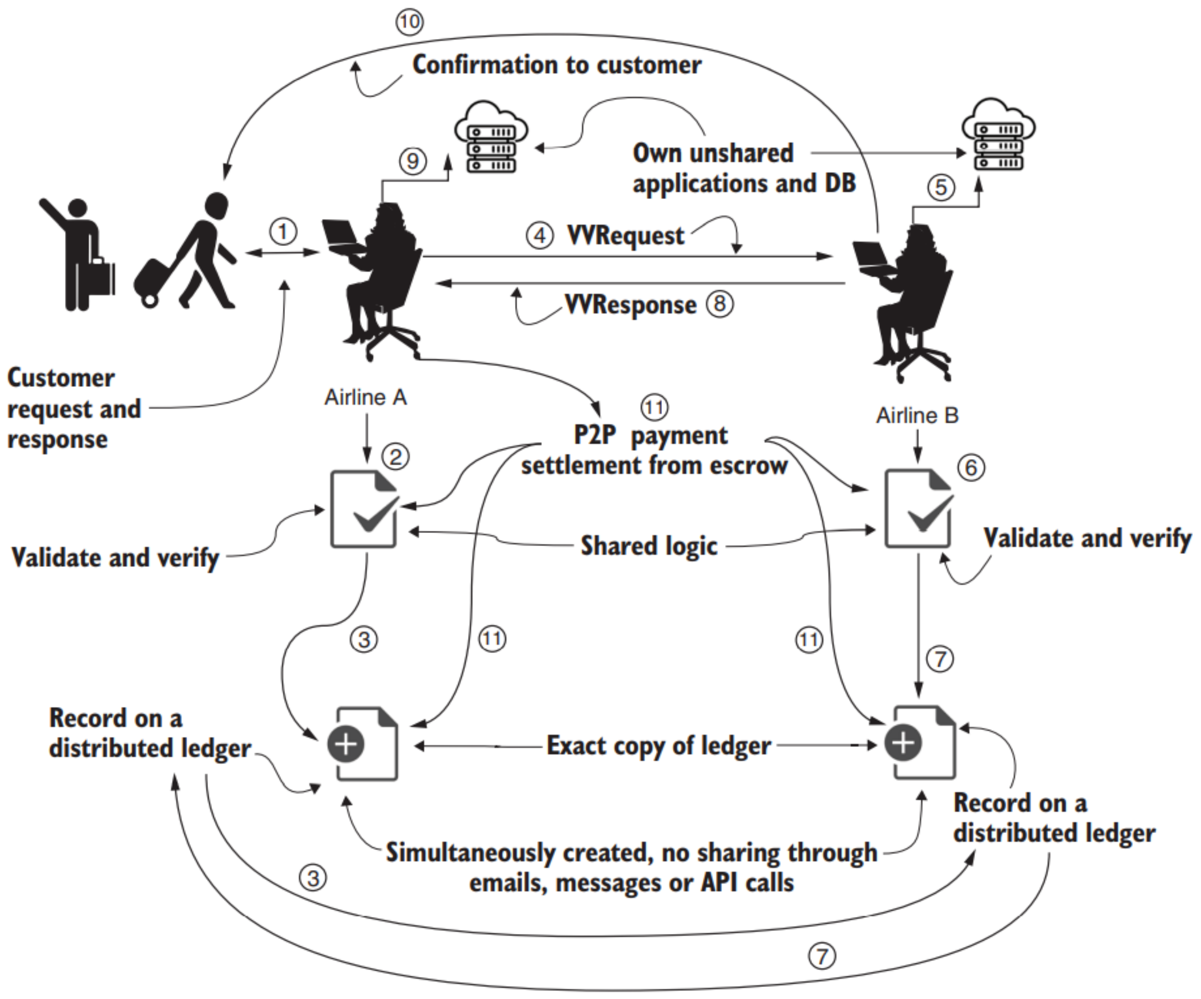
Figure 2.4 Class diagram versus contract diagram templates

UML class diagram as a guide for the design of the solution to the counter problem. A class diagram defines the various structural elements of the solution. It draws upon the items discovered in the previous two steps (creating the use case diagram and digital asset analysis). The typical UML class diagram of traditional object-oriented programming (OOP) shown on the left side of figure 2.4 has three components:
ϒ Name of the class
ϒ Data definitions
ϒ Function definition
The smart contract diagram on the right side of figure 2.4 has one additional compo nent: the rules for accessing the functions and data. This component distinguishes a smart contract diagram from a traditional class diagram. Design principle 4 deals with contract diagrams.

5. **What makes a blockchain contract smart?**

Here are some cool features of a smart contract that make it smart. A smart contract is equivalent to any participant in a blockchain network because it has
ϒ A name
ϒ An address
ϒ A cryptocurrency balance (ether, in this case)
ϒ Built-in features to send and receive cryptocurrency (ether, in this case)
ϒ Data and functions
ϒ Built-in features to receive messages and invoke functions
ϒ The ability to reason out the execution of a function
These aspects distinguish and set apart a smart contract from a regular piece of code. A smart contract is indeed smart and different.

6. Explain in details Operations of participants in a decentralized airline syste.


10
Confirmation to customer
9
Own unshared applications and DB
5
1
4 VVRequest
VVResponse 8
Customer request and response
Airline A
11
P2P payment settlement from escrow
Airline B
2
6
Validate and verify
Shared logic
Validate and verify
3
11
11
7
Record on a distributed ledger
Exact copy of ledger
Record on a distributed ledger
Simultaneously created, no sharing through emails, messages or API calls
3
7

1. A customer initiates a change of flight seat that they hold on airline A.
2. An agent or application at airline A verifies and validates the request through smart contract logic shared among the ASK consortium members.
3. Once verified, the request Tx is confirmed and recorded in a distributed immutable ledger. Now everyone in the consortium knows that a legitimate request has been made.
4. In the simplest design, an agent at airline A sends the verified and validated request (VVRequest) to airline B. (Alternatively, we could use a broadcast model in which many airlines get the request, and any one of them could respond.)
5. An agent or application at airline B checks the airline's database to check for availability
6. An agent at airline B responds through shared smart contract logic that verifies and validates the common interests and shared rules of the consortium
7. Once verified, the response Tx is confirmed and recorded in a distributed immutable ledger. Now everyone in the consortium knows that a response has been sent.
8. Airline B sends the response (indicated by VVResponse) to the agent at airline A.
9. Airline A updates its database, noting that a change has been made.
10. An agent at airline B sends the customer the information for the flight seats and other details. (Note that airline B holds its data assets and transfers them directly to its known customer, not to airline A.)

11. Payments are settled through peer-to-peer Txs, using the escrow or deposit that participating airlines hold in their shared smart contract. The payment settle ment can be embedded in other suitable operations in the system but will be handled by the shared smart contract and recorded in the ledger. This settle ment is automatically carried out by the smart contract logic

7. Explain in brief (i) Peer participants, (ii) data assets, (iii) roles, (iv) rules, and (v)transactions

USERS:

You first identify the users of the system as peer participants. This term is used to emphasize that they interact with one another in a peer-to-peer fashion, without any intermediaries.

ASSETS:

The data assets are the flight seats and the funds held by the peer participants. We learned in chapter 1 that one of the fundamental tenets of a decentralized system is that peer participants—not an intermediary—hold their own assets.

ROLES

The following are the roles:

- Agents acting on behalf of the airlines can enroll or self-register by using the register() function with an escrow/deposit;
- Agents (of members only) can request flight seats.
- Agents acting on behalf of the airlines can check their centralized databases for availability and reply.
- Peers settle the payment between themselves if seats are available.
- The chairperson of the consortium has the sole authority to unregister members and return leftover deposits.

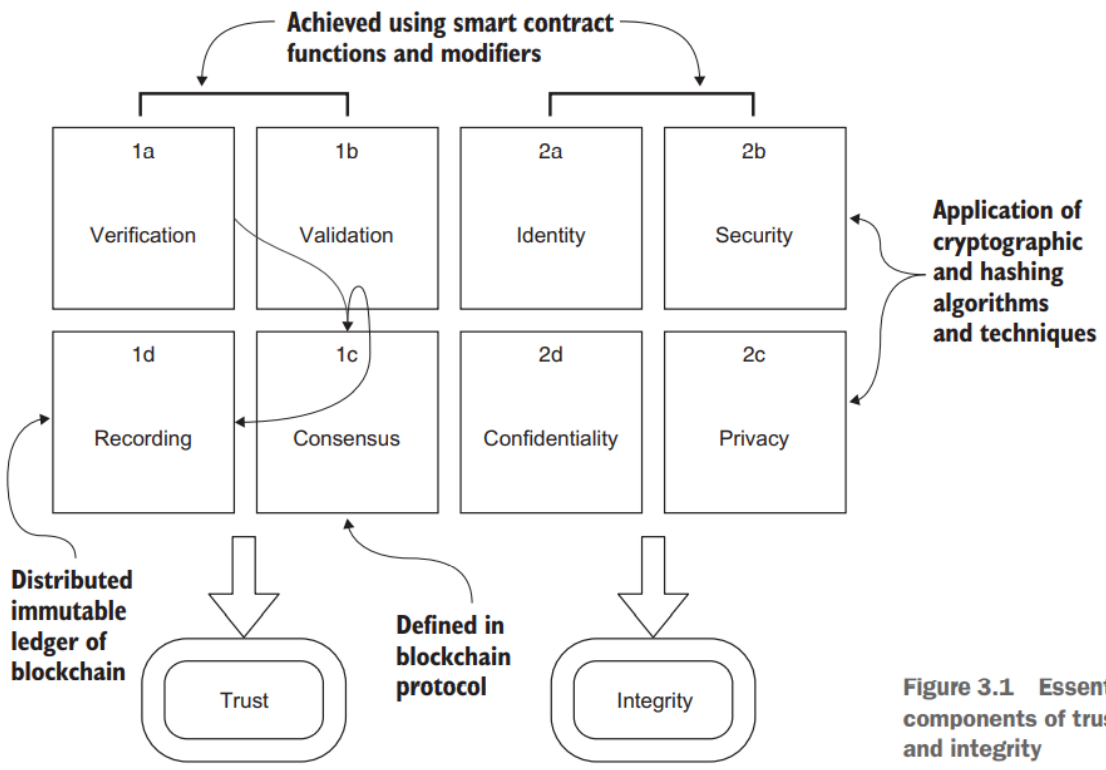
TRANSACTIONS

A typical process of buying flight seats may involve many operations and interactions with various subsystems, such as databases. Let's refer to operations that need to be verified, validated, confirmed, and recorded by all parties as the transactions, or simply Txs.

RULES

You specify the rules or conditions, using a control structure called a modifier. Solidity language provides the modifier feature to code the rules. ϒ Modifiers are used for specifying who can access the functions and who can access the data, and also for validating data uniformly.

8. What are the Essential components of trust and integrity.



Achieved using smart contract functions and modifiers
1a Verification
1b Validation
2a Identity
2b Security
1d Recording
1c Consensus
2d Confidentiality
2c Privacy
Application of cryptographic and hashing algorithms and techniques
Distributed immutable ledger of blockchain
Defined in blockchain protocol
Trust
Integrity

Figure 3.1 Essential components of trust and integrity

Trust is a measure of confidence in the credibility of a peer partic ipant in a system. Trust in a blockchain-based system is established by verifica tion and validation of relevant participant data and transactions, and by immutable recording of appropriate information done with the consensus of the stakeholders.

Integrity, in the context of blockchain, means ensuring the secu rity and privacy of data and confidentiality of transaction

9. Write a short note on Finite state machine diagram.

To represent system dynamics, you'll use a UML finite state machine, or FSM diagram. The FSM is well founded in formal computer science and mathematics, but it is also a versatile UML design diagram. It is an important diagram because it represents the various state changes a smart contract goes through that are dependent on time and other conditions. Often, the conditions and rules are based on various phases of a real-world contract or process, which brings us to design principle 5. DESIGN PRINCIPLE 5 Use a finite state machine UML diagram to represent sys tem dynamics such as state transitions within a smart contract.

10. In FUNCTION DETAILS what are the five functions, including the constructor.

There are five functions, including the constructor:
ϒ constructor()—The constructor function is called when the smart contract is deployed. The account number that deploys the contract is that of the chairperson. The constructor takes the number of proposals to be voted on as a parameter. It initializes the data elements and the state of the voting phase (to Regs from Init).

ϒ changeState()—This function changes the state of the voting to the correct phase. It can be executed only by the chairperson, and the parameter value has to be in the correct order (1, 2, 3). Execute this function from the chairperson account's address before transitioning to register(), vote(), and reqWinner() for the first time. The statement if (x < state) revert(); works only for simple state advancement. This basic version of state change is improved to a generic version in chapter 4.

ϒ register()—This function should be executed only by the chairperson account; otherwise, it will revert and won't be executed. It will also revert if the voted Boolean variable is true and if the state is not Phase.Regs.

ϒ vote()—This function can execute only during the voting phase (Phase.Vote). This rule is enforced by the modifier (rule) validPhase(Phase.Vote). You can observe the validation of the "one person–one vote" rule and the proposal number. (When the voting period ends, the state is changed to Phase.Done by the chairperson.)

ϒ reqWinner()—This function counts the votes and identifies the winning proposal by its number. It executes the counting every time it's called. During testing, this is okay because you might call the function once or twice, but for production, you may want to optimize it. (Also, in future designs, you'll move this function off-chain or out of the smart contract code.) Note that this function is a "view" function, so it's not recorded on the chain.

11. Explain in details Architectural model of a blockchain network for two processes.

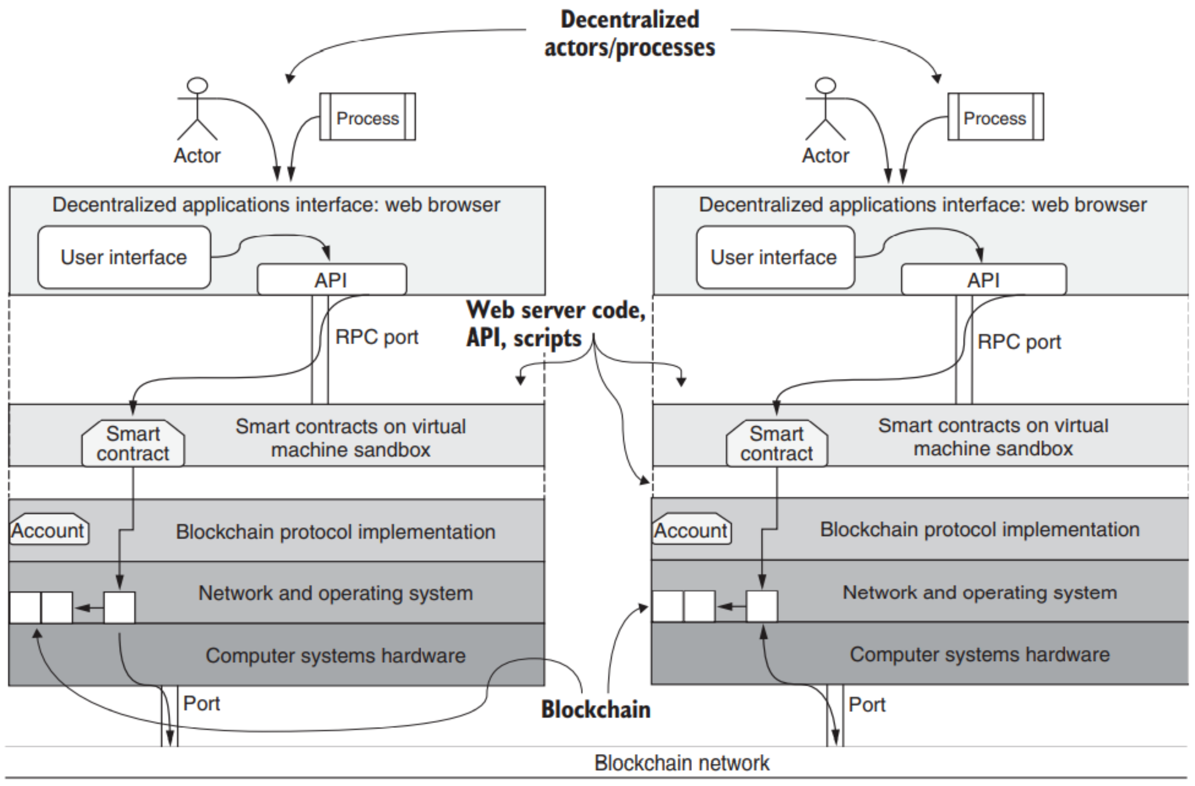

Decentralized actors/processes
Actor
Process
Actor
Process
Decentralized applications interface: web browser
User interface
API
Decentralized applications interface: web browser
User interface
API
RPC port
Web server code, API, scripts
RPC port
Smart contract
Smart contracts on virtual machine sandbox
Smart contract
Smart contracts on virtual machine sandbox
Account
Blockchain protocol implementation
Account
Blockchain protocol implementation
Network and operating system
Network and operating system
Computer systems hardware
Computer systems hardware
Port
Blockchain
Port
Blockchain network

Figure 4.2 Architectural model of a blockchain network

Starting at the top, the users (actors) or processes acting on behalf of users invoke the UI functions. These functions use web application software and blockchain APIs to connect to smart contract functions. Txs representing the smart contract function invocations are recorded on the blockchain. (Note that only some of the necessary Txs will be recorded.) You can follow the operational flow in a node from an actor to the consis tent blockchain recording on both nodes via the blockchain network. This figure also illustrates how a blockchain-based Dapp is not a standalone application but is dependent on its host operating system's file system, ports, and network capabilities.

12. Explain the Dapp stack with diagram.

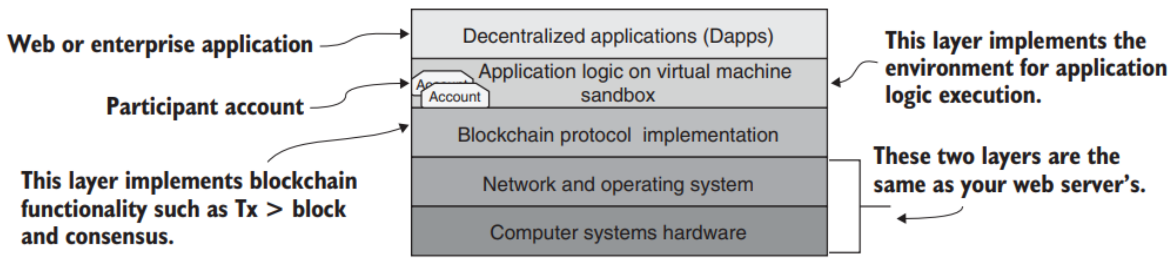

Web or enterprise application
Participant account
This layer implements blockchain functionality such as Tx > block and consensus.
Decentralized applications (Dapps)
Application logic on virtual machine sandbox
Account
Blockchain protocol implementation
Network and operating system
Computer systems hardware
This layer implements the environment for application logic execution.
These two layers are the same as your web server's.

Figure 4.1 Dapp stack

13. Explain with a diagram the Dapp development layers.

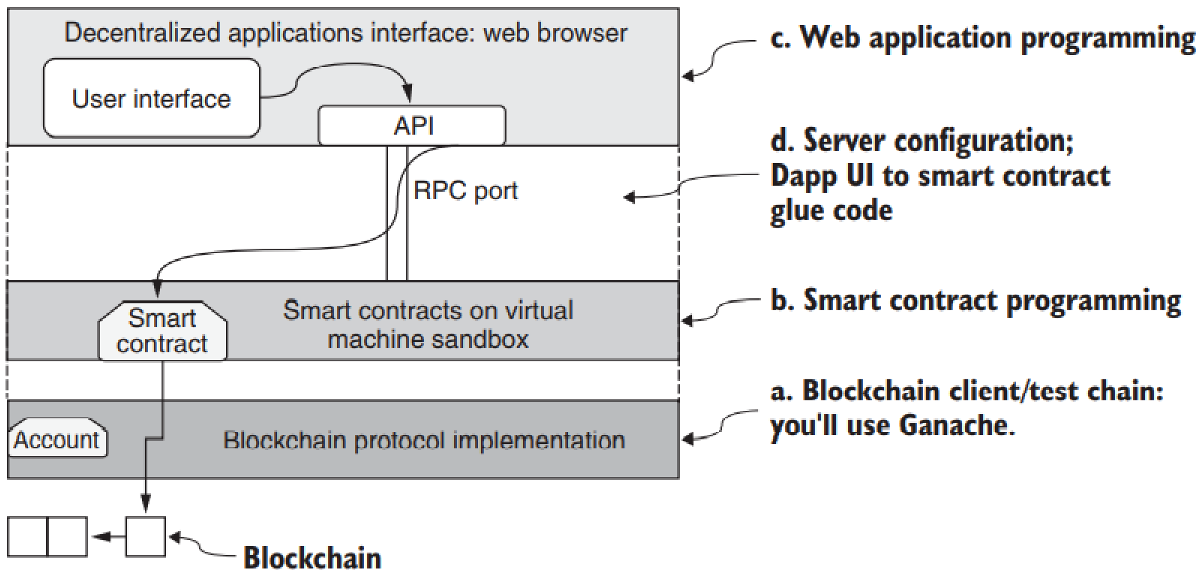

Decentralized applications interface: web browser
User interface
API
RPC port
Smart contract
Smart contracts on virtual machine sandbox
Account
Blockchain protocol implementation
Blockchain
c. Web application programming
d. Server configuration; Dapp UI to smart contract glue code
b. Smart contract programming
a. Blockchain client/test chain: you'll use Ganache.

Figure 4.3 Dapp development layers

14. Explain the Web files and folders in the src directory.

The src directory contains
ϒ The usual artifacts for a web page (CSS, fonts, images, JavaScript).

ϒ The landing page for the web application (index.html).

ϒ proposals.json, which holds details about the proposals being voted on. (Images for the proposals are in the images subdirectory.)

ϒ app.js, the glue code connecting the web server layer and the smart contract layer.

You can review the files for the web app by unzipping src.zip in the codebase for this chapter and cloning it into the ballot-app folder, which contains index.js package.json src.
The src directory contains the source for the web application part. Communication from the web client to the blockchain server is through JSON over RPC. Let's focus on analyzing app.js, which contains the handlers for the stimuli invoking the smart contract functions. I'll discuss this app.js

code later for two reasons: this part will be different for different smart contracts and Dapp, and you'll have a better idea of the role of app.js after you run the Dapp and interact with it.

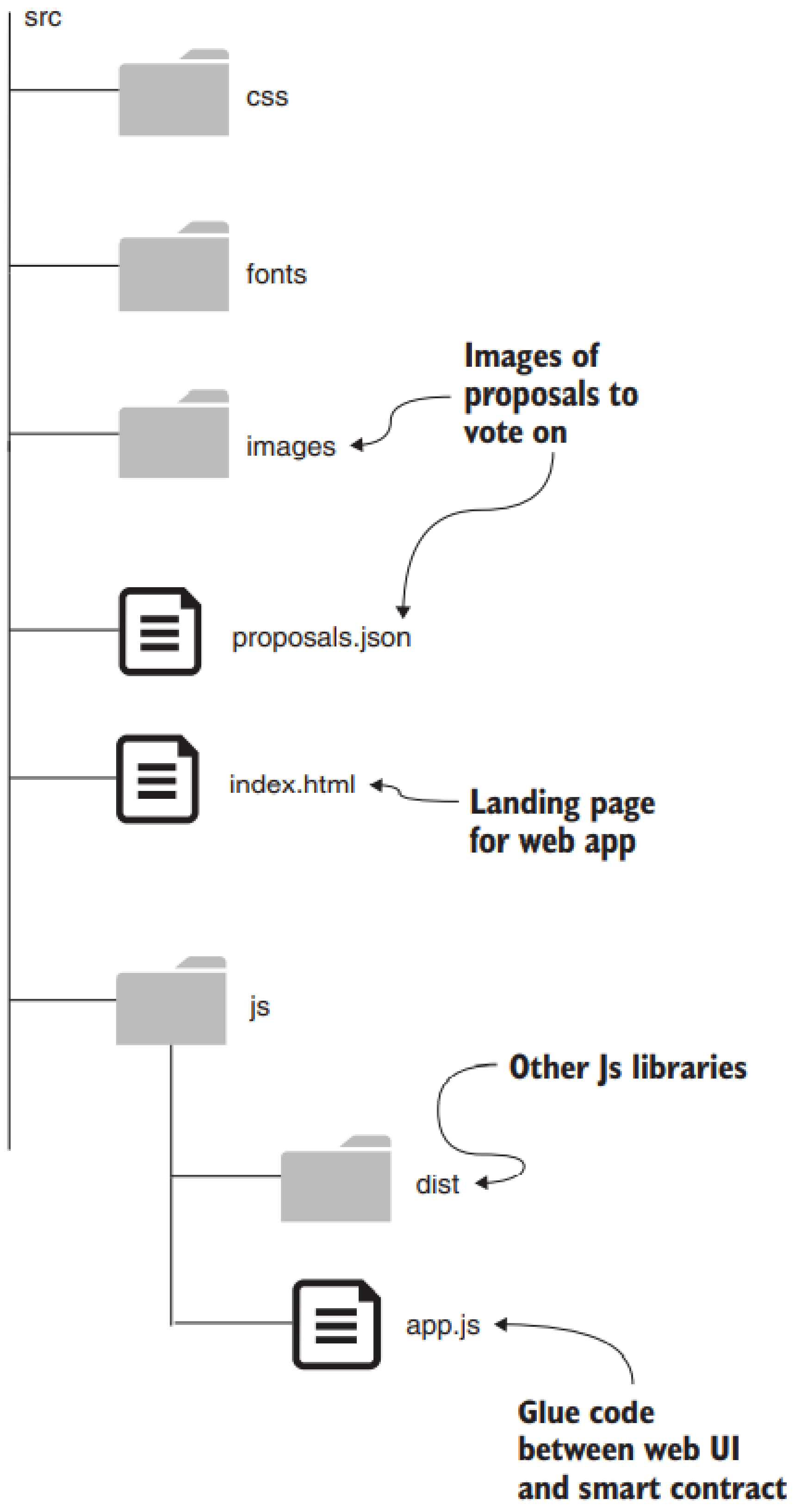
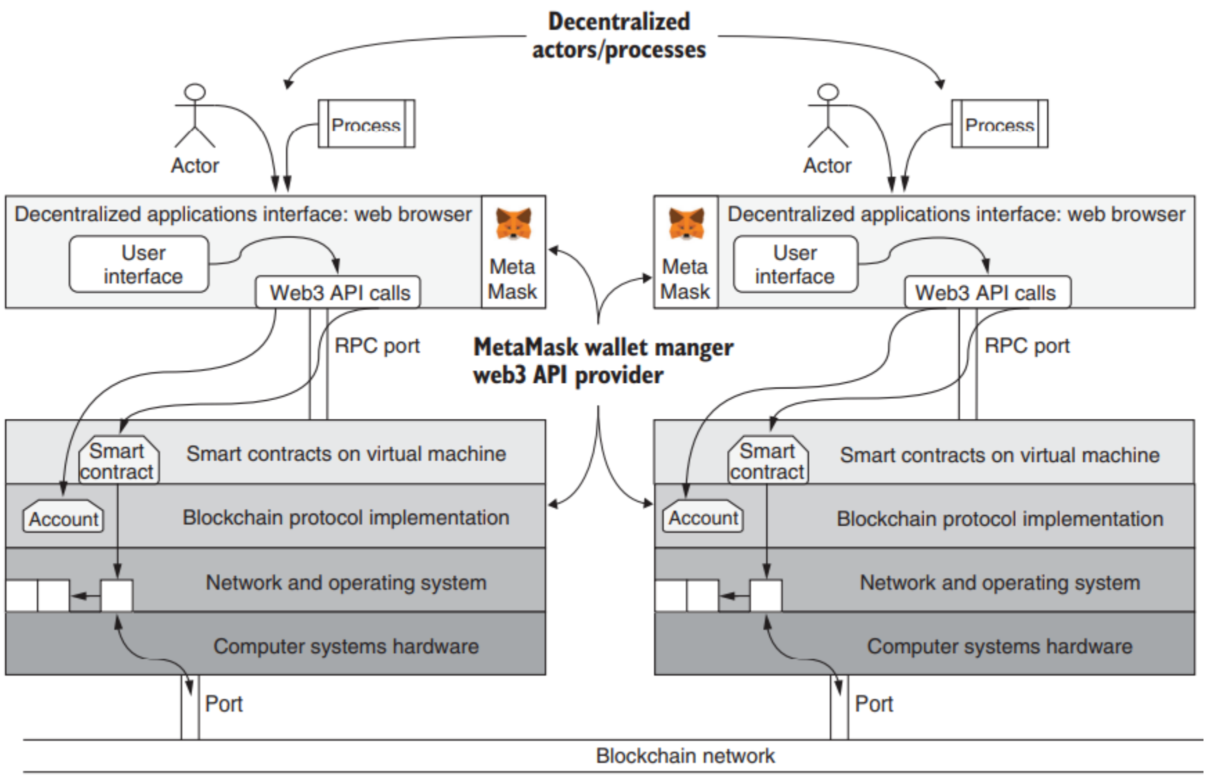


Figure 4.7 Web files and folders in the src directory

15. **Explain in bried Dapp Tx flow from a user via web API, RPC port, smart contract and blockchain with a diagram.**

The figure shows two nodes, each with its users interacting with the web client and to the blockchain node through smart contracts and contributing to the construction of the distributed ledger of the blockchain



Decentralized actors/processes
Actor
Process
Actor
Process
Decentralized applications interface: web browser
User interface
Web3 API calls
Meta Mask
Meta Mask
Decentralized applications interface: web browser
User interface
Web3 API calls
RPC port
MetaMask wallet manger web3 API provider
RPC port
Smart contract
Smart contracts on virtual machine
Account
Blockchain protocol implementation
Network and operating system
Computer systems hardware
Smart contract
Smart contracts on virtual machine
Account
Blockchain protocol implementation
Network and operating system
Computer systems hardware
Port
Port
Blockchain network

Figure 4.8 Dapp Tx flow from a user via web API, RPC port, smart contract and blockchain

16. **Explain with a diagram the Elements of trust and integrity.**

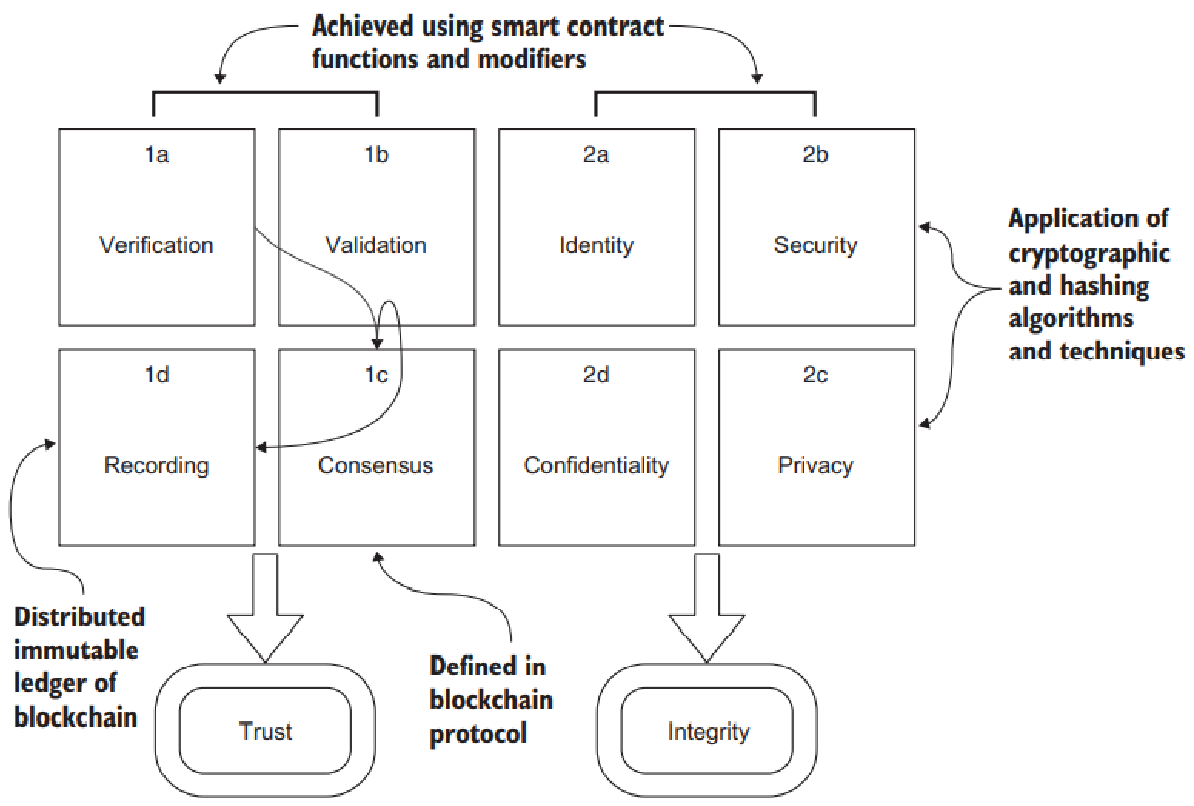
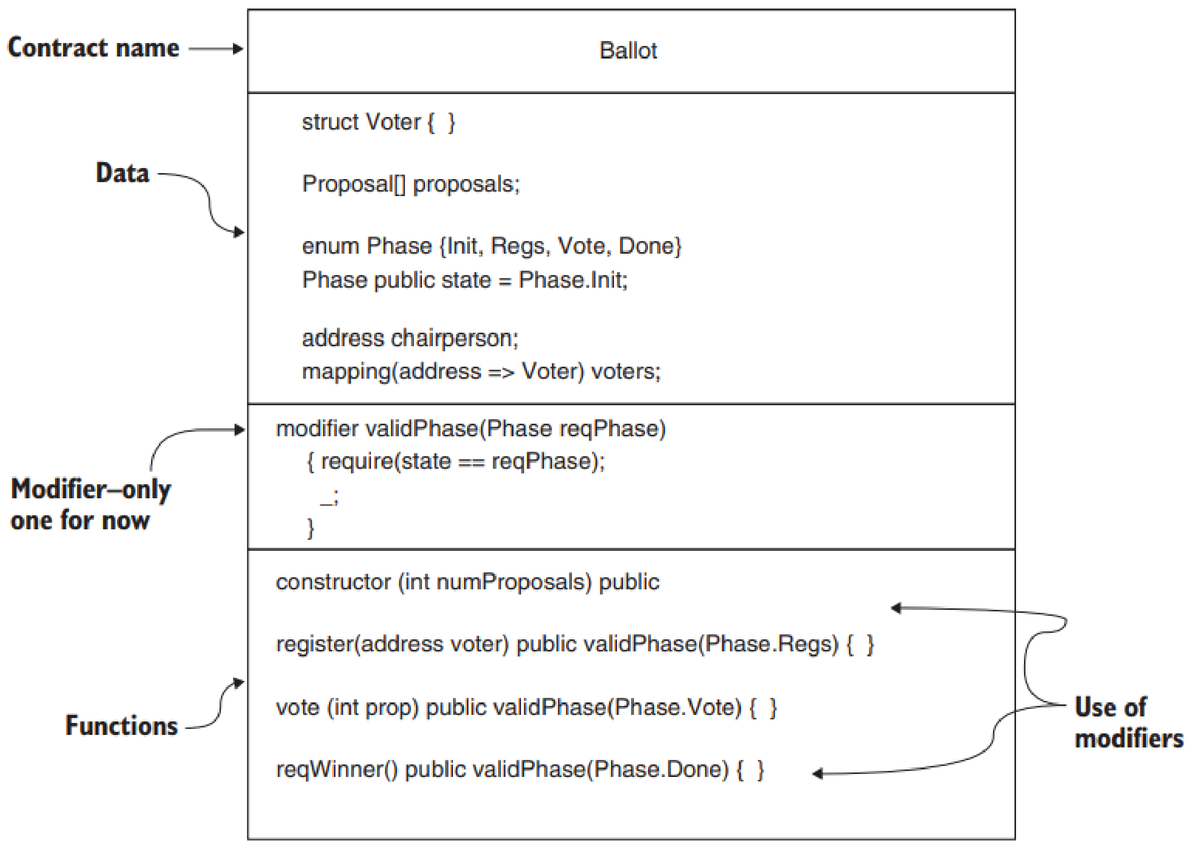

Achieved using smart contract functions and modifiers
1a
Verification
1b
Validation
2a
Identity
2b
Security
Application of cryptographic and hashing algorithms and techniques
1d
Recording
1c
Consensus
2d
Confidentiality
2c
Privacy
Distributed immutable ledger of blockchain
Trust
Defined in blockchain protocol
Integrity

Figure 5.1 Elements of trust and integrity

17. Explain with a diagram Contract diagram including modifiers.

In the contract diagram shown in figure 3.6, you can see the definition of one modifier, validPhase, in the modifier box after the data definition. In this case, only one example of a modifier is defined to help you understand the modifier feature.

Digital democracy problem



```
Ballot

struct Voter { }

Proposal[] proposals;

enum Phase {Init, Regs, Vote, Done}
Phase public state = Phase.Init;

address chairperson;
mapping(address => Voter) voters;

modifier validPhase(Phase reqPhase)
  { require(state == reqPhase);
    _;
  }

constructor (int numProposals) public

register(address voter) public validPhase(Phase.Regs) { }

vote (int prop) public validPhase(Phase.Vote) { }

reqWinner() public validPhase(Phase.Done) { }
```

Figure 3.6 Ballot contract diagram

Note that the modifier validPhase has a parameter Phase reqPhase. In the functions box of the contract diagram, you see the repeated use of the validPhase modifier in the headers of three functions. Observe that the validPhase modifier is called with three different actual parameters—Regs, Vote, and Done—from the headers of the various functions, which illustrates the flexibility and reusability of the modifier. Before each function, the modifier is applied and executed with the actual parameter value. Inside the modifier, this actual parameter is compared with the current state of the voting process. If it does not match the state at the time the function is called, the function call is reverted, and it is not executed or recorded on the blockchain. This validation is the role of the modifier. Now you can proceed to complete the Solidity code for the Ballot contract based on the details specified in the contract diagram.

18. Explain with a diagram, Blockchain application stack and layers