UNIT- IV

Ethereum Block chain Implementation: Introduction, Tuna Fish Tracking Use Case, Ethereum Ecosystem, Ethereum Development, Ethereum Tool Stack, Ethereum Virtual Machine, Smart Contract Programming, Integrated Development Environment, Truffle Framework, Ganache, UnitTesting, Ethereum Accounts, My Ether Wallet, Ethereum Networks/Environments, Infura, Etherscan, Ethereum Clients, Decentralized Application, Metamask, Tuna Fish Use Case Implementation, Open Zeppelin Contracts.

Tuna Fish Tracking Use Case

Introduction: -

We will discuss about the **Tuna Fish supply chain use case** and how this solution can be architected. We have chosen a multi-actor supply chain for tracking tuna fish.

One of the persistent problems in tuna fishing industry is illegal, unreported, unethical, and unregulated fishing in the Pacific region.

Consumers are now increasingly looking for transparency, ethical sourcing, and full traceability to make sure that the fish they purchase does not come from any illegal fisheries.

A simple QR code scan should reveal the story of tuna fish to its consumers. The data collected from RFID tracking and IoT devices can be inserted into the blockchain.

This gives details on when and where the fish was caught processing temperatures, result of quality checks by auditors, etc.

i)Use Case Description:

As very simple level, the use case will explain the flow of events and activities that need to happen. The following are the steps that have been identified:

1.Workflow:

(a) Fishermen capture tuna and send it to the supplier.
(b) Supplier will send the tuna received from the fishermen to the fish-processing unit.
(c) The processed fish is then sent to the distributor with the help of a transporter.
(d) The distributor now splits and distributes the fish to various retailers.
(e) The retailer sells the fish to restaurants and supermarkets.
(f) Customers buy tuna from the restaurants and supermarkets.
(g) Flow of tuna through the various parties can be inspected by the regulator as well as customers.

2 Benefits for players:

(a)Customers get to make informed decision regarding the food they are consuming, such as if it is coming from a source that fits their ethical framework. Customers are happy to pay more for this transparency.
(b) Law abiding and ethical fishermen get the right price and do not need to depend on unethical. practices for the sake of needing to make ends meet.
(c) Regulators get a full view of the supply chain and can trap any incorrect entry early.
(d) All other participants benefit through the value generated and extra money spent by customers.

Only a few main scenarios have been captured to depict how to approach the design of blockchain applications. The actual solution will need more significant details

ii)Blockchain Relevance Evaluation Framework:

We will perform the fitness assessment in this section. Let us go through the questions listed in the assessment. These are not the final set of criteria but a good set of primary criteria to be considered.

1. **Does use case involve sharing assets/valuables/data between multiple participants?**
   Yes. Multiple participants who are not confined to a single organization are involved. The fish is the asset that is shared between various participants.

2. **Is there impact due to sharing/hiding information between participants?**
   Yes, intermediaries are involved. The fish does not reach the restaurant or the supermarkets directly. They are passed through intermediaries. In this example, the intermediaries can also be accommodated in the blockchain. It should be noted that not all blockchain use cases focus on eliminating intermediaries. Some of the use cases may still involve intermediaries, but the reliance on intermediaries is reduced.

3. **Does the solution require shared write access?**
   Yes. Fish is the asset that moves through the supply chain, that is, the ownership of the fish is changed as it moves through the supply chain.

4. **Can the solution work without delete?**
   Yes. As of today, the tuna supply chain does not mandate delete of data. Also, delete of change of hands will not help build the desired transparency.

5. **Is it required to store large amount of non-transactional data?**
   No. The properties of fish can be generally digitized and stored on the blockchain. Any related documents shall be stored on a central database by the needed parties outside the blockchain.

6. **Is it required that only a very small group or an entity needs all the control?**
   Yes. The parties participating in the supply chain can control the supply chain functionality.

7. **Does use case involve high-performance (millisecond) transactions?**
   No. This process spans through various participants over a period of few days to few weeks and does not demand any high-performance transactions.

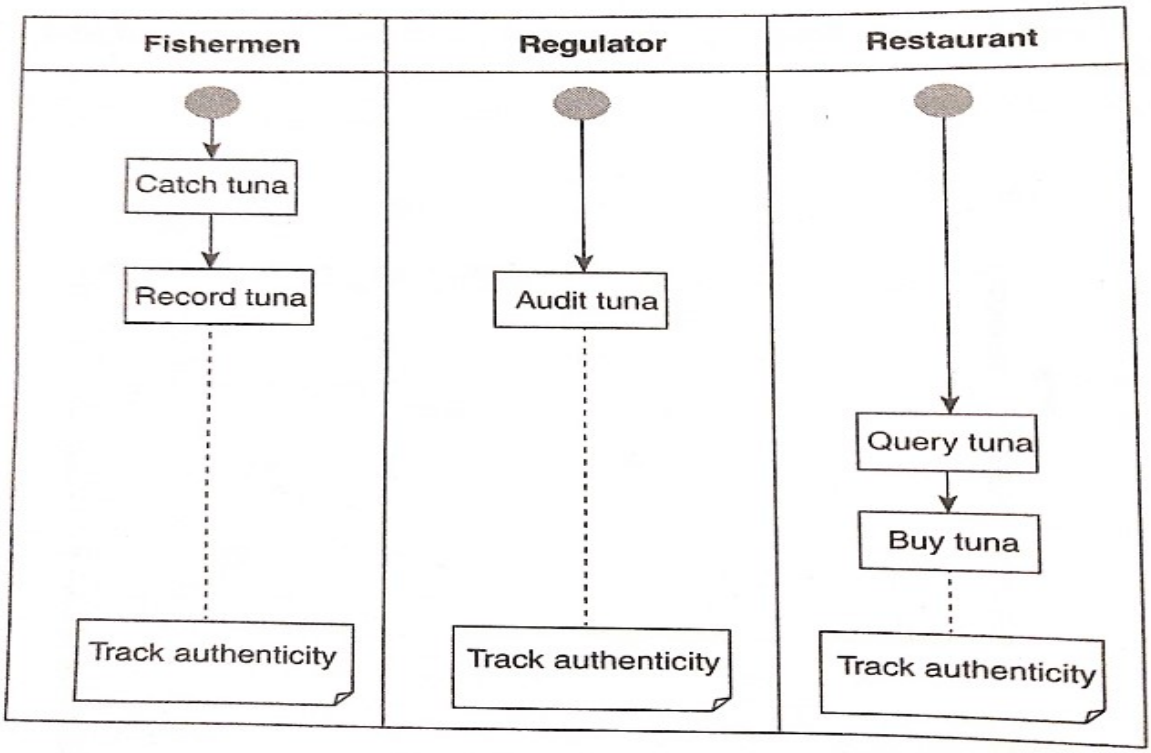
Now, we know that the assessment revealed that it is a very good use case for blockchain.

iii)Actors, Roles and Permissions:

a) Producer (Fishermen)
b) Supplier
c) Fish-Processing Unit
d)Transporter
e) Distributor
f) Retailer
g) Restaurant
h) Supermarket
i) Restaurant Customers
j) Regulator

Now we will capture the various activities that shall be performed by each of the actors. For the sake of simplicity, only free actors are chosen for the subsequence section-fishermen, regulator, and restaurant.
However, the solution approach and concepts can be extended to more number of actors as well.
We will be utilizing Unified modeling language (UML) diagrams to capture these details.

iv)Activity Diagram:



Activity diagram.

v)Sequence Diagram:

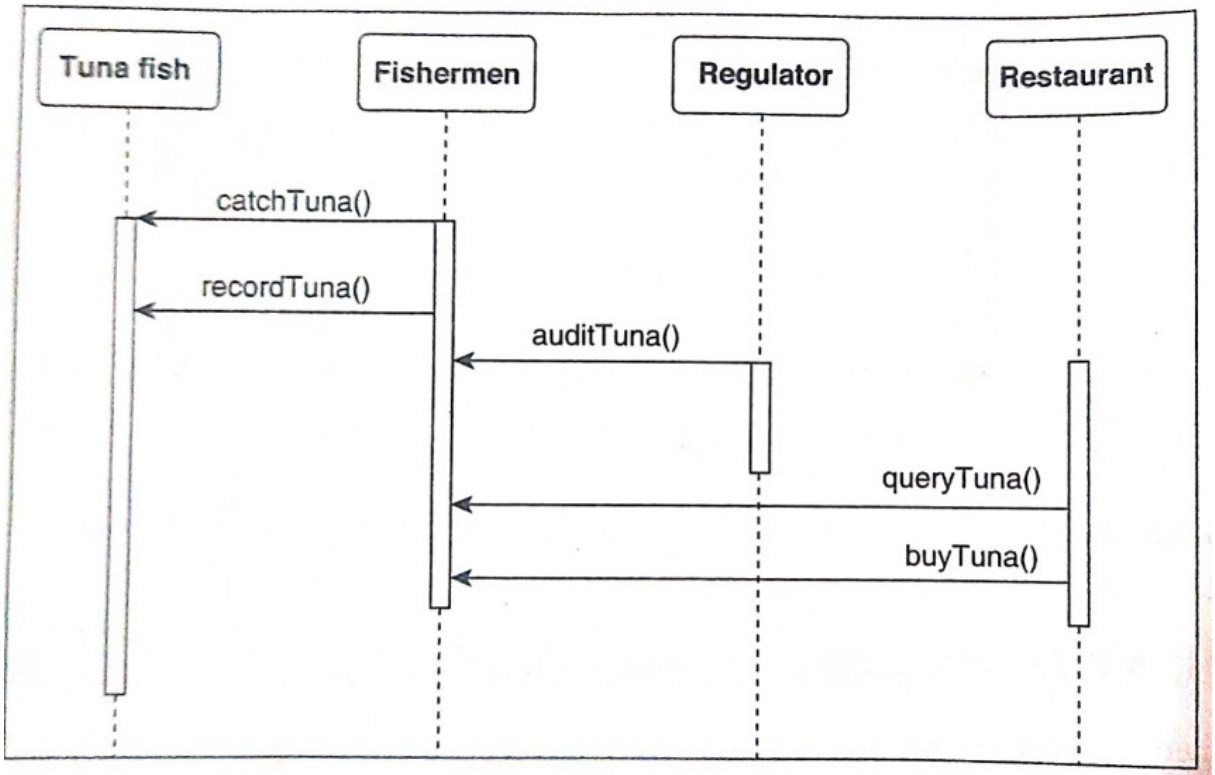


Figure 5.6 Sequence diagram.

Vi)State Diagram

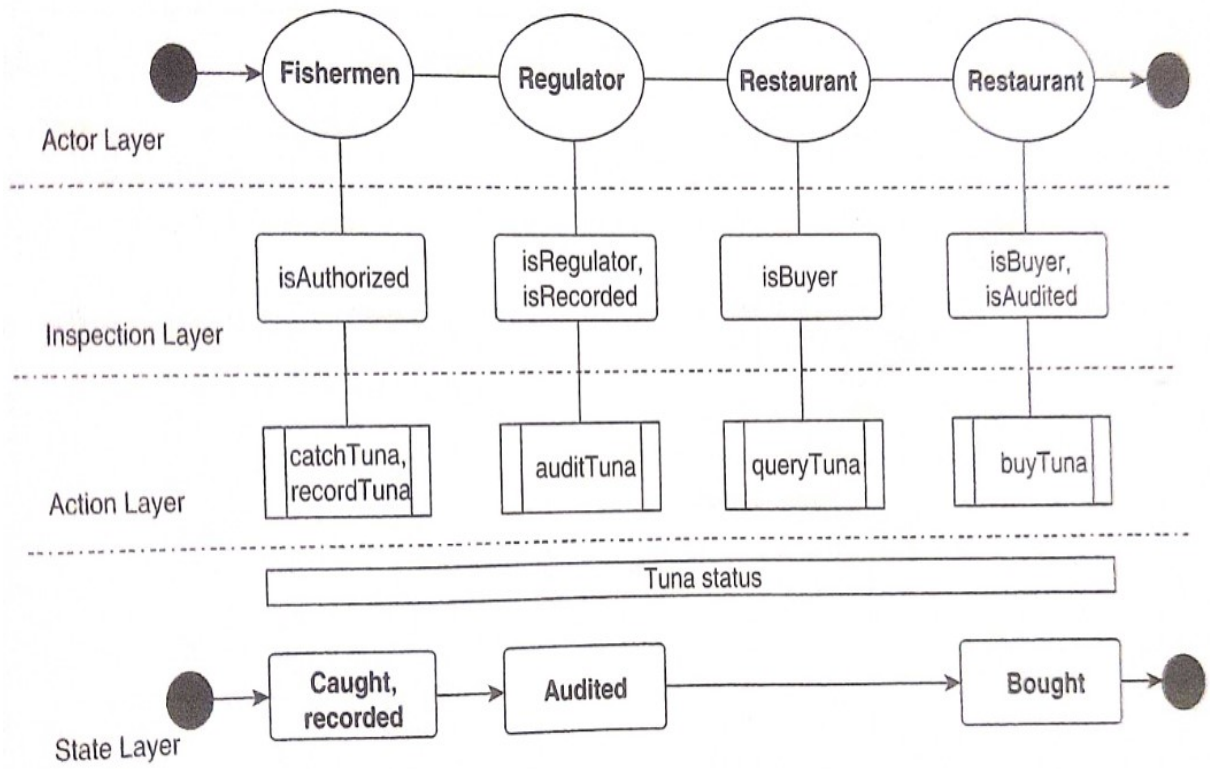


Figure 5.7 State diagram.

For further understanding, the **state diagram** is split into multiple layers. The bottom layer of Figure 5.7 marked as State Layer shows hoy the state of tuna fish changes during its journey. This gets recorded on blockchain via smart contracts.

The layer above this is represented as **Action Layer**, which shows the corresponding action that results in that particular state change.

This is accompanied by another layer known as **Inspection Layer**, which performs some simple checks to ensure that an actor is authorized to perform particular action and the current state of tuna is valid to perform that action.

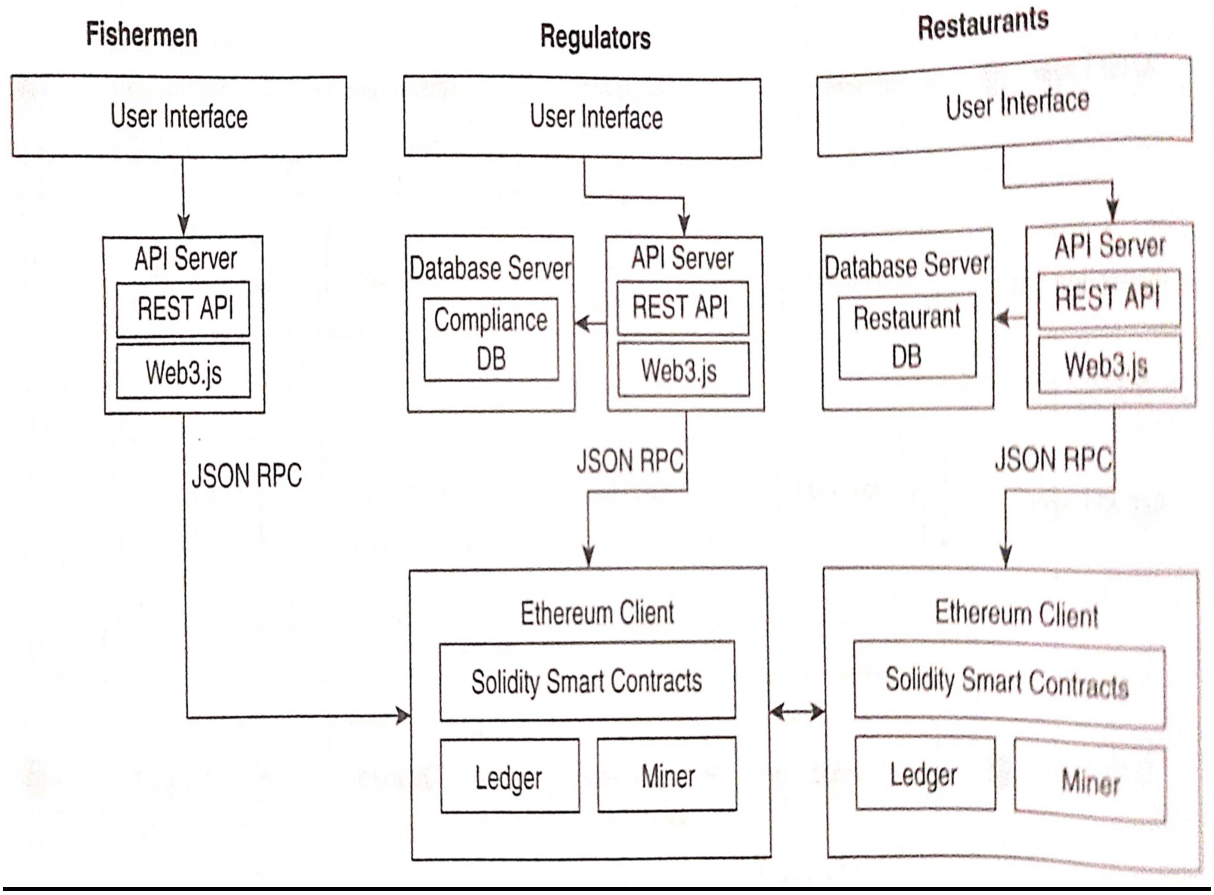
For example when a regulator wants to carry **AuditTuna** activity, the following checks occur:

1.isRegulator: This will check whether the user is a Regulator.

2.isRecorded: This will ensure that the current status is "Recorded", which means that the metadata related to tuna fish is recorded on blockchain.

The topmost layer of the state diagram represented in Figure 5.7 as **Actor Layer**, shows different actors and the sequence in which they perform activities.

vii)Solution Architecture:

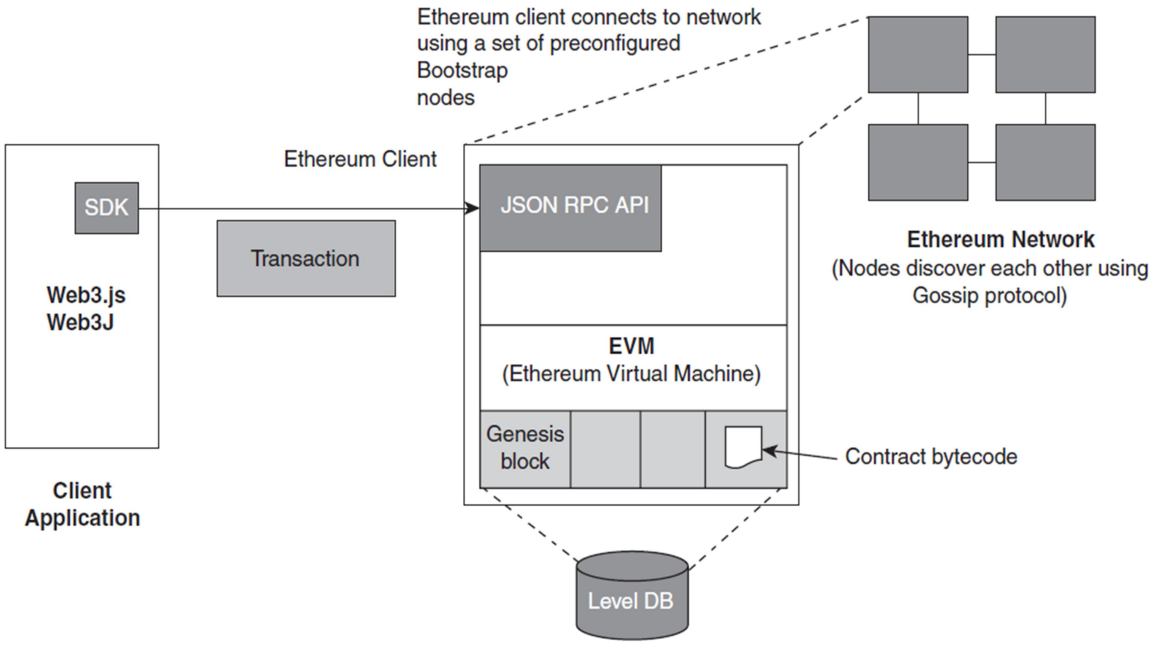


Ethereum Ecosystem

Introduction: -

The heart of Ethereum architecture is the Ethereum Virtual Machine (EVM), which provides runtime environment for Ethereum blockchain.

The Ethereum ledger itself is implemented using LevelDb that stores all the transactions initiated from a client application, which is also known as decentralized application (DApp).



To execute smart contracts, Ethereum utilizes something called **Ethereum virtual machine** (EVM), which is a kind of isolated sandbox for the smart contract code running on blockchain. This restriction where smart contract cannot access network, file system or other node assets directly provides secure and independent environment for logic execution.

On Ethereum, smart contracts are typically coded in a language called **Solidity,** complied using an EVM compiler and tested and deployed utilizing provided tool set like the Truffle. The tool set includes capability of providing a sample node with unlocked accounts, so that developers can utilize their own machines to develop, deploy and test code before it is ready to be tested on the real network.

DApps and oracles can interact with the network utilizing a library called web3.js for cryptocurrency trading or smart contracts.

In Ethereum platform implementation, every blockchain node will be an Ethereum client. Various Ethereum clients available in the market are Geth, Aleth, Trinity, Parity, EthereumJS, Ethereum), etc.

The most widely used Ethereum client is Geth. Every Ethereum client consists of miner functionality that participates in the consensus process. Client applications connect to the Ethereum Client using an Ethereum library that does the heavy lifting of handling the communication between them.

This allows developers to focus more on implementing the functionality rather than the communication between client application and Ethereum client.

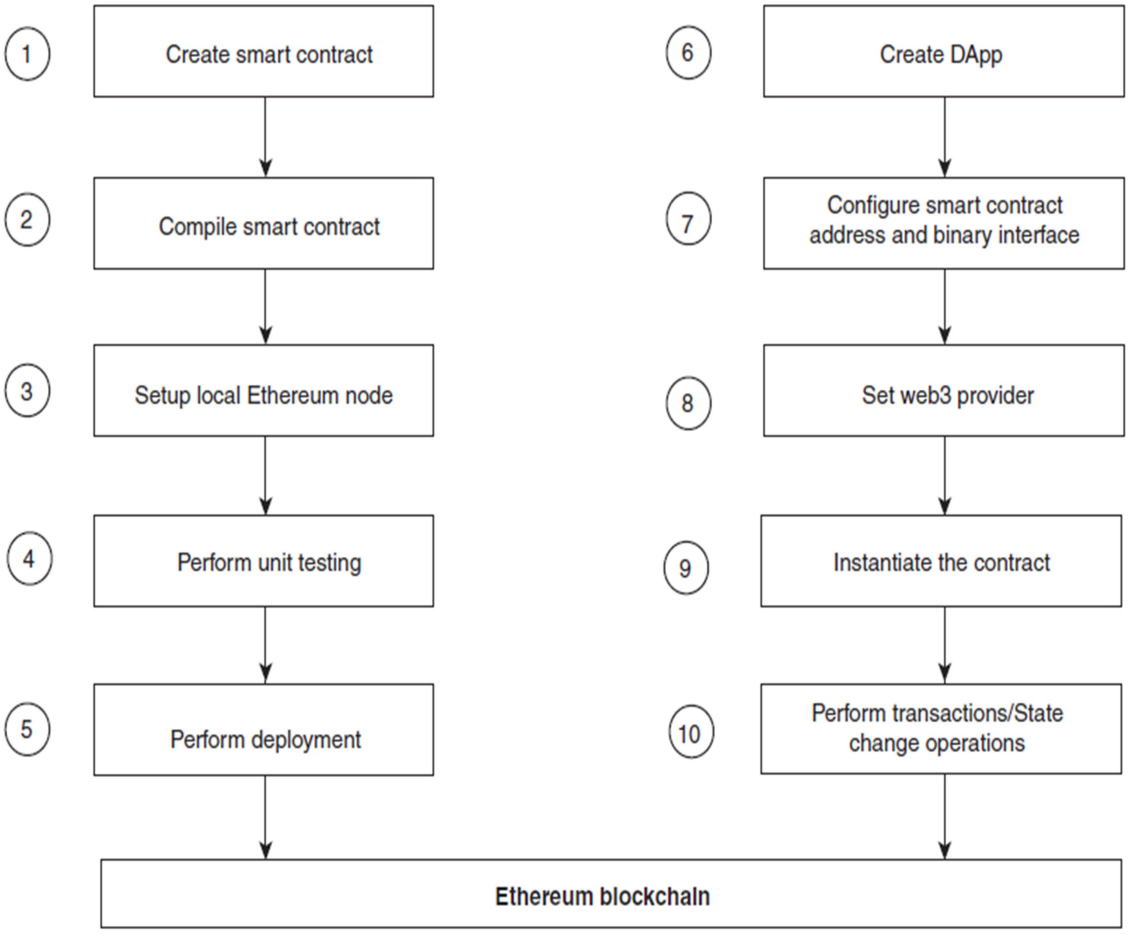
Ethereum Development

Introduction:-

Implementing any blockchain use case on Ethereum platform essentially involves creating smart contracts and allowing external applications to interact with them.

At a very basic level, creating an Ethereum solution in development environment involves the following 10 steps.

1. Create smart contract using high-level Ethereum programming language.
2. Compile smart contract to bytecode to be executed on Ethereum runtime environment.
3. Set up the Ethereum blockchain node.
4. Perform unit testing.
5. Deploy smart contract onto Ethereum node.
6. Create a frontend DApp to interact with smart contract.
7. Get the smart contract address and binary interface outputs from step 5.
8. Set up and assign a web3 library.
9. Invoke smart contract via remote procedure call (RPC) using web3 modules. 10. Perform transactions.



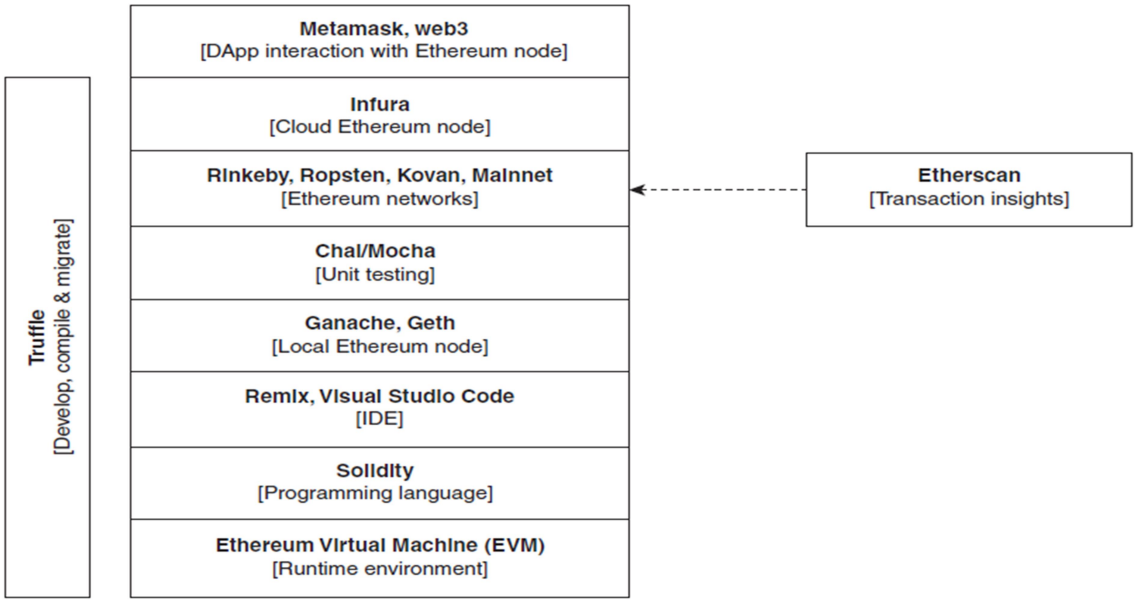
Once this solution is developed and tested on local environment, it can be deployed to higher environ- ments.

There are various tools available which assist Ethereum developers with the above-mentioned steps. The subsequent sections deal with some important tools within the Ethereum ecosystem.

Ethereum Tool Stack

Introduction:-

Ethereum tool stack contains a variety of tools that offer different features – from providing runtime environment to interaction with DApps.



Ethereum tool stack.

Ethereum Virtual Machine

Introduction:-

Ethereum smart contract is basically a program that runs on Ethereum blockchain. This program requires a runtime like any other program. This runtime is provided by a component known as EVM which forms the heart of Ethereum blockchain network.

EVM can be viewed as analogous to JVM, which provides runtime for Java applications. However, EVM has no access to system and network resources.

Executing instructions on EVM:-

EVM is a quasi-Turing complete machine. Executing instructions on EVM incurs some cost known as **gas**. Ethereum platform has a built-in token known as ether.

Gas in this context means some fraction of ether which needs to be supplied along with other inputs while executing a transaction or smart contract on Ethereum. This is similar to fuel (gas/petrol/ diesel) that is required to run motor vehicles.

A vehicle runs as long as fuel is available. In the same manner, code execution on Ethereum continues till the availability of gas. Hence, EVM is known as a quasi-Turing complete machine due to the dependency on gas availability for execution.

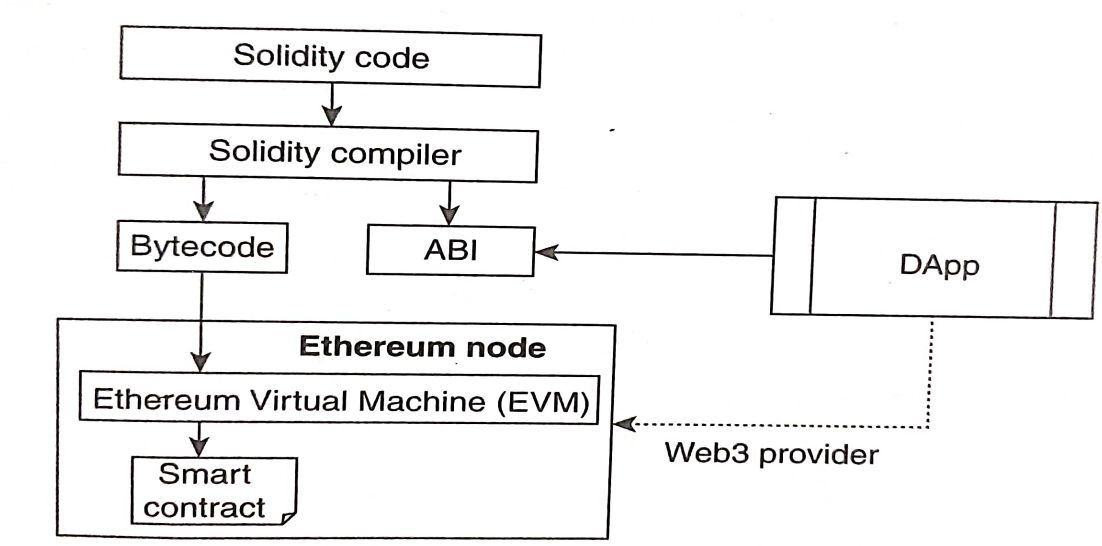
Also, note that gas offers protection against infinite loop scenarios in smart contract programming.

Interaction between Smart Contract, EVM, and DApp:-

When an external application such as DApp wants to interact with the smart contract, it needs some details about the contract such as the function signatures and properties exposed by it.

This is similar to how a distributed application interacts with an external web service using its service description language.

The Ethereum language compiler generates a human readable interface known as application binary interface (ABI), which contains all the necessary details needed by a DApp to interact with the smart contract using an external helper library such as web3.



Interaction between smart contract, EVM, and DApp.

Smart Contract Programming

Introduction:-

Ethereum supports different languages such as **Solidity, Bamboo, Vyper, Fl**int, etc., to write smart contracts.

Solidity is the most popular and widely used language for Ethereum smart contract programming.

Solidity is a high-level language influenced by C++, Python, and JavaScript.
It resembles a lot with JavaScript and supports multiple data types, conditional statements, and loops.

The language supports object-oriented features such as
-inheritance,
-class,
-interfaces,
-functions, and properties.

It also supports libraries and complex user-defined types.

Layout of a Solidity Source File:-

```
pragma solidity <version>
import <path to another solidity file>
contract <name>{
      //variables
      //function modifiers
      //events
      //functions
}
```

Example:

```
pragma solidity 0.5.2
import "../HelperContract.sol";
contract HelloWorld
  {
    string msg="Hellow World";
    function sayHello() public returns (string)
          {
           return msg;
          }
  }
```

Date Types in Solidity:-

- Bool
- Int /uint
- address
- Byte
- Bytes
- String
- Array
- Mapping
- Struct
- Enum

Expressions and Control Structures:-

Solidity supports control structures supported in most of the languages — if, else, while, do, for, break, continue, return.

The syntax would be similar to JavaScript or C language.

Contract Instance and Constructor Function: -

Similar to instantiating a class in other object-oriented programming languages, Solidity contracts can be instantiated using the keyword new.

Example: obj= new HelloWorld();

Solidity contract can have a Constructor Function, which is executed automatically when an object of the Solidity Contract is created. Constructor Functions are created using constructor keyword.

Example:

```
pragma solidity >=0.5.2
import "../HelperContract.sol";
contract mycontract1
  {
   string msg;
   constructor(string arg)
         {
          msg=arg;
         }
   function sayHello() public returns (string)
         {
           return msg;
         }
}
```

Modifiers:

- Basically, modifiers are reusable code segments that are used to change the behavior of functions.
- When a modifier is applied on a function, they get executed first before the actual function. In this manner, one can automatically check a condition before executing the function.
- The syntax to declare modifiers is using the *modifier* keyword followed by the modifier name.

Example:

```
pragma solidity >=0.5.2
contract mycontract2
  {
   string msg;
   constructor(string arg)
         {
          msg=arg;
         }
   modifier check
        {
         …..
        }
   function sample() public check returns (string)
         {
          return msg;
         }
  }
```

Integrated Development Environment

Introduction:-

Integrated Development Environment (IDE) is a set of tools that help the developers in writing, compiling, executing, and debugging code.

There are many IDEs that support Ethereum smart contract development.
Two of the most widely used IDEs for Ethereum development are:

-Remix
-Visual Studio Code (VS Code)

Remix: -

- Remix can be accessed online from a browser or from a locally installed copy.
- Developers can write Solidity code in Remix editor, which offers features such as color coding for keywords, syntax highlighting, compilation and execution of Solidity Code.

- Remix can be directly launched through the URL **https://remix.ethereum.org/**
- Remix provides many Options / facilities such as:
  -File Explorer
  -Plugin Manager
  -Editor Window
  -Solidity Compiler
  -Debugger Window
  - Contract Deployment

File Explorer: which allows you to create a new Program File or open an existing File.

Plugin Manager: Which allow you to manage packages.

Editor Window: It allows you to type your program code

Solidity Compiler: Smart contract Programs are compiled using this.

Debugger Window: provides facility to see errors or outputs

Contract Deployment: provides facility to perform deployment of the project

Visual Studio Code (VS Code): -

- VS Code is an open source code editor developed by Microsoft. It is available on Windows, Mac, and Linux.

- It is highly customizable and offers various features such as Syntax highlighting, snippets, intelligence, Code refactoring, Integrated terminal, GitHub integration, etc.

- The VS Code terminal can be used to download any no. of packages required during smart contract creation

Truffle Framework

Introduction:-

- The Truffle framework is a suite of tools that assist in setting the development environment, testing smart contract and also serves as asset pipeline for Ethereum.
- It aims to perform a lot of heavy lifting and make Ethereum development easier.

- It offers a variety of features, which are as follows:
  1.Smart contract compilation, linking, and deployment.
  2.Executing automated unit test cases using Mocha and Chai.
  3.Deploying to public and private networks.
  4.Scriptable deployment and migrations framework.
  5.Network management for deploying many public and private networks.
  6.Interactive console for direct contract communication.
  7.External script runner that executes scripts within a Truffle environment.

Truffle Installation

- Truffle is comes with a variety of tools suited for different needs.
- It can be installed using the following command form a node console or VS Code Terminal.
  npm install –g truffle

After installing Truffle, developers can start writing Solidity code from scratch or use a boilerplate template provided by Truffle. These boilerplate codes are provided by a tool known as **Truffle Box**.

- They contain sample Solidity contracts, other helpful modules and libraries, frontend views and more way up to complete example DApps. In this way, Truffle integrates EVM, smart contract programming, migration, and frontend DApp technologies.

- To download a boilerplate code from Truffle Box, run the command
  truffle unbox <box-name>
  from VS Code terminal. Usually, this template includes a basic project structure, smart contract, and the code for user interface.

Default directory structure created during initialization is as follows:

1. contracts/: This contains the Solidity source files for smart contracts. There is an important contract in here known as Migrations.sol, which will be described subsequently.

2 migrations/: This is used to handle smart contract deployments. To keep track of the changes, a special smart contract named migration will be used.

3. test/: This may contain Solidity and JavaScript tests for smart contracts.

4 truffle-config.js: This is a Truffle configuration file. This file may have a different name, truffle.js, on Mac/Linux.

Ganache

Introduction:-

- Ganache is a personal blockchain for running Ethereum smart contracts. It comes in two flavors:
  -command line interface (CLI) and
  -graphical user interface (GUI

It provides 10 built-n Ethereum accounts with 100 ether each. These can be used for development and testing purposes with local blockchain.

Upon opening Ganache, it shows a list of accounts created, the mnemonic used to create those accounts, account balance and related transactions.

Ganache GUI client can be installed from their website.

Unit Testing

Introduction:-

- Whenever any piece of code is written, it is best practice to write automated unit test cases also.
- The Truffle ecosystem supports Chai and Monacha frameworks to write unit test cases for Solidity smart contracts.

To Execute Unit Test Cases, we have to Run the Command **truffle test** from VS Code terminal (or simply **test** inside **truffle develop** interactive console)

- After unit testing on local blockchain network, the smart contract can be migrated to other networks.
- When using Ganache to work locally, Ethereum accounts with some Ethers that are needed to migrate and perform transactions were automatically generated.

Ethereum Accounts

Introduction:-

- An Ethereum account is a combination of public–private key pair created using Elliptic Curve Digital Signature Algorithm (ECDSA).
- This account is represented by a hexadecimal value known as address.
- The address is obtained by computing keccak-256 hash of the public key and taking rightmost 20 bytes of that hash.
- An Ethereum address would look like this:
  0x9f8702f95eed51a9418fdda5c8d6ad06355e70a6.

- There are multiple options to create new Ethereum accounts:
  -geth console,
  -Mist wallet, or
  -MyEtherWallet.

- The basic difference between an account and a wallet is that an account is a public–private key pair, whereas a wallet hosts account and also provides some advanced features such as transaction log, multi-signature options, etc.

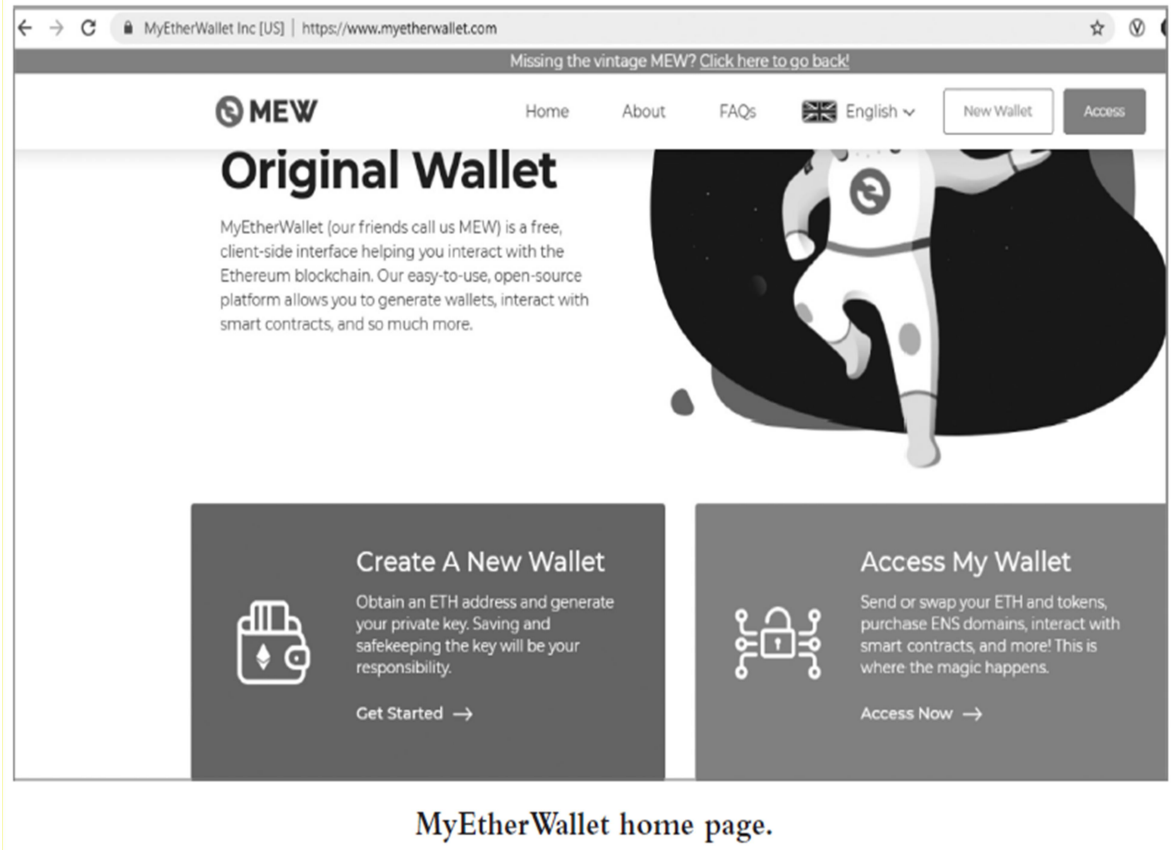
- One of the most commonly used application to create Ethereum wallets and associated accounts is MyEtherWallet (MEW).

MyEtherWallet (MEW)

Introduction:-

MEW is an easy-to-use, open-source platform that allows generation of Ethereum wallet address.

No data about the generated wallet address gets stored on their platform and account owners have complete control of the addresses they own.



MyEtherWallet home page.

Ethereum Networks / Environments

Introduction:-

- Ethereum blockchain network nodes validate and record transactions. They also contain runtime to execute smart contracts. Currently, Ethereum has the following two types of networks:
  -Mainnet
  -Testnet:

Mainnet: -

Mainnet is short for "Main Network" and is the original and functional blockchain where actual transactions take place.

Any native currency used in this network possesses real economic value. The NetworkID for Mainnet is "1".

Testnet: -

Testnet is short for "Test Network" and is mainly used for development and testing purpose. It is supposedly an exact replica of the original blockchain, with the same technology, software, and functionalities.

The only difference is that transactions on Testnet are simulated (or "fake") and any native currency used does not possess any real value outside.

- Ethereum blockchain nodes for Testnet and Mainnet can be run on a developer machine as well.
- However, there is a cloud service named **Infura** that provides this functionality and reduces the overhead for developers to run and maintain these nodes.

Infura

Introduction:-

- Infura is a hosted Ethereum node cluster that lets the users run their application without requiring them to set up their own Ethereum node or wallet.

- Infura is a platform as a service (PaaS) product for Ethereum platform.
  It channels an average of more than 6 billion JSON–RPC requests per day on Ethereum network and is one of the essential pillars of Ethereum ecosystem.

- First, developers should register with Infura. This allows creation of new project and generates API endpoint URL and a ProjectID known as Infura key.

- This key should be kept private and will be used in the Truffle config file for migrations.

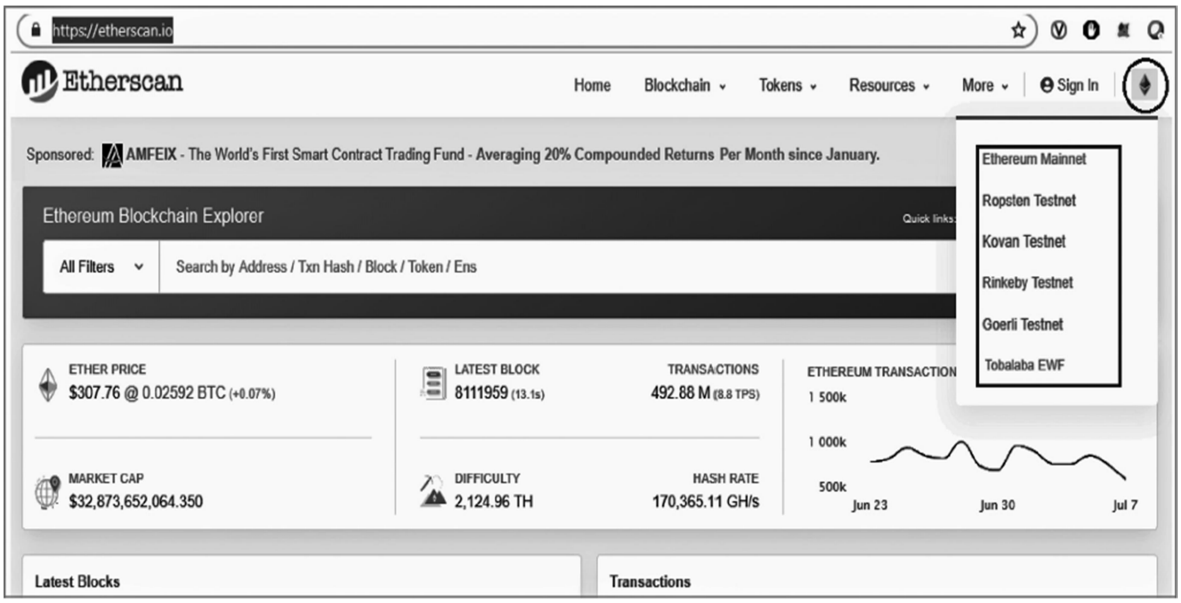
- The Infura endpoint URL will be different for each Ethereum network. Infura cannot sign transactions on behalf of a user during any interaction with blockchain.

- However, Truffle can do this using a library known as HDWalletProvider.

Etherscan

Introduction:-

- Etherscan is a frontend interface that allows users to explore and search Ethereum blockchain for transactions, addresses, and other activities.
- It also offers APIs and analytics platform for Ethereum blockchain.
- Etherscan will be very helpful to visualize transaction details while deploying a smart contract to Testnet and also during its testing by invoking from a DApp.



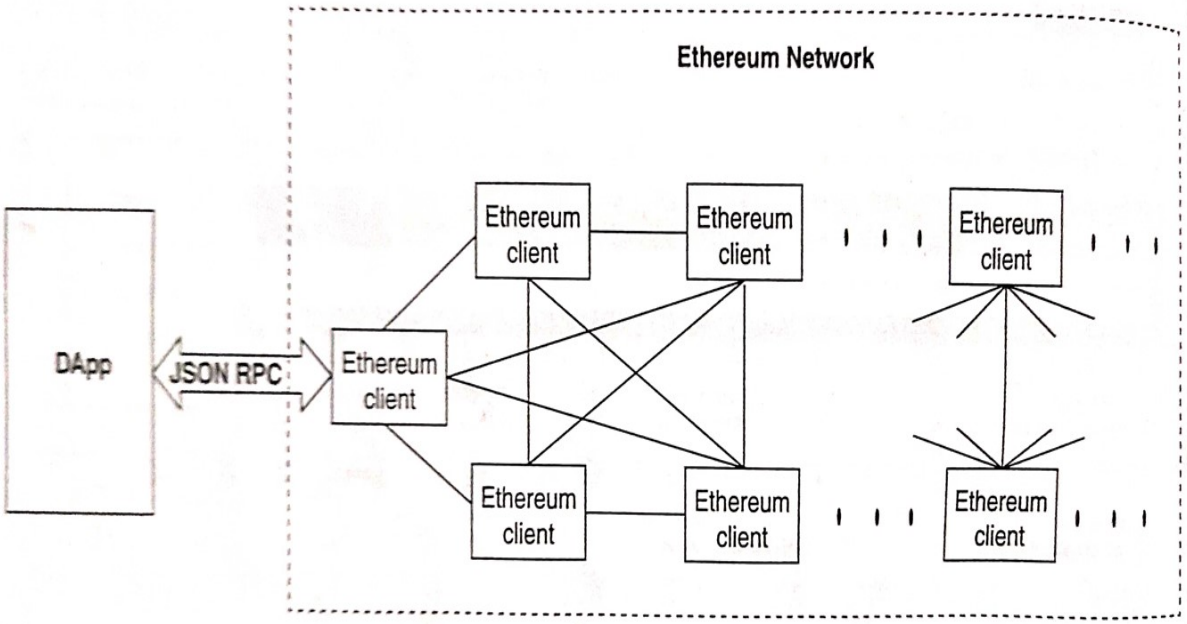
Etherscan home page.

Ethereum Clients

Introduction:-

- An Ethereum client is any node that can run Ethereum blockchain – verify transactions, execute smart contracts, and mine blocks.
- Official CLI reference implementations of Ethereum client are as follows:
  - **Eth:** C++ client of the webthree project. It was formerly known as cpp-ethereum.
  - **Geth:** Golang client of the go-ethereum project.
  - **Pyethapp:** Python client of the pyethereum project.
- Graphical clients available by the Ethereum core developers are as follows:
  - **Mist:** Works on top of eth or geth and aims to be a DApp browser. It currently implements the ethereum-wallet-dapp.
  - **Alethzero:** This is being deprecated.
- There are some non-official clients that implement the following yellow paper specification:
  parity – Rust client,
  ethereumj – Java client,
  ethereumjs-vm – EVM in JavaScript,
  ethereumH – Haskell client,
  ruby-ethereum – Ruby client,
  node-blockchain-server – simple JavaScript server.

An application can interact with Ethereum client using JSON-RPC as depicted here. However, this adds a lot of development overhead.



DApp interaction with Ethereum network.

Various libraries are available to do the heavy lifting of this communication with Ethereum blockchain.

This allows developers to focus more on implementing the functionality rather than the communication aspects. Table 5.3 lists some of the prominent libraries.

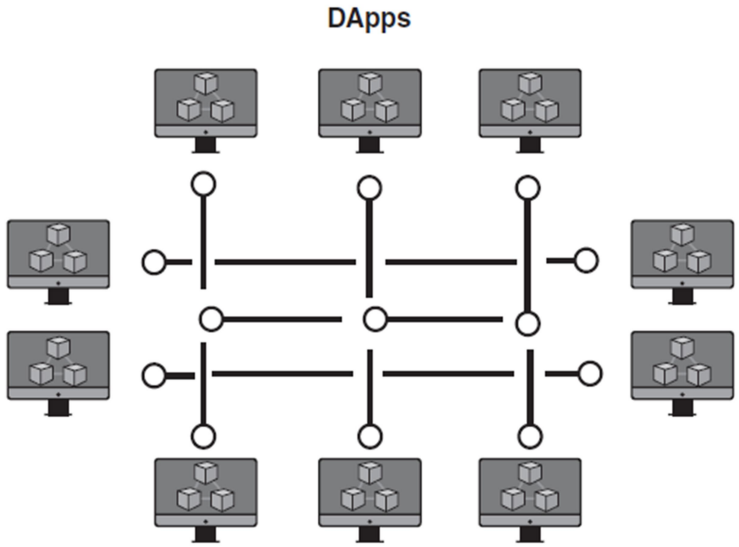
Table 5.3 List of prominent JSON RPC libraries

Library	Language	Notes
web3.js	JavaScript	This is available as a node module on npm. Project source: [Link 13]
web3j	Java	This is a lightweight Java library for interacting with Ethereum clients. Currently, go-ethereum and Parity clients are supported. Project source: [Link 14]
Nethereum	C#.Net	This is available as NuGet package and supports go-ethereum, cpp-ethereum, and Parity clients. Project source: [Link 15]
ethereum-ruby	Ruby	This is a Ruby JSON–RPC wrapper to interact with Ethereum node. Currently, go-ethereum client is supported. Project source: [Link 16]

Decentralized Application

Introduction:-

- Decentralized application (DApp) is an application that runs on a peer-to-peer network of computers.
- This network can be blockchain or non-blockchain.
- Our focus will be on DApp that runs on a blockchain network.



DApp with its backend running on decentralized peer-to-peer network.

Metamask

Introduction:-

- Metamask is a browser extension that lets an application owner or user run DApp without being part of the Ethereum network as an Ethereum node.
- Instead, it allows users to connect to an Ethereum node on Infura and run smart contracts on that node.
- Metamask can manage Ethereum accounts within its wallet and allows the user to send and receive Ethers through a DApp of interest.
- Following are the steps to set up Metamask.
  - Install Metamask browser plugin.
  - Create an account and back up the secret pass phrase.
  - Click on the Metamask browser plugin and then on the My Accounts icon at the top right.
  - Metamask allows users either to create a new Ethereum account or import an existing one.

Tuna Fish Use Case Implementation

Introduction: -

We will discuss about the **Tuna Fish supply chain use case** and how this solution can be architected. We have chosen a multi-actor supply chain for tracking tuna fish.

A simple QR code scan should reveal the story of tuna fish to its consumers. The data collected from RFID tracking and IoT devices can be inserted into the blockchain.

This gives details on when and where the fish was caught processing temperatures, result of quality checks by auditors, etc.

i)Use Case Description:

As very simple level, the use case will explain the flow of events and activities that need to happen. The following are the steps that have been identified:

1.Workflow:

(a) Fishermen capture tuna and send it to the supplier.
(b) Supplier will send the tuna received from the fishermen to the fish-processing unit.
(c) The processed fish is then sent to the distributor with the help of a transporter.
(d) The distributor now splits and distributes the fish to various retailers.
(e) The retailer sells the fish to restaurants and supermarkets.
(f) Customers buy tuna from the restaurants and supermarkets.
(g) Flow of tuna through the various parties can be inspected by the regulator as well as customers.

2 Benefits for players:

(a)Customers get to make informed decision regarding the food they are consuming, such as if it is coming from a source that fits their ethical framework. Customers are happy to pay more for this transparency.
(b) Law abiding and ethical fishermen get the right price and do not need to depend on unethical. practices for the sake of needing to make ends meet.
(c) Regulators get a full view of the supply chain and can trap any incorrect entry early.
(d) All other participants benefit through the value generated and extra money spent by customers.

Only a few main scenarios have been captured to depict how to approach the design of blockchain applications. The actual solution will need more significant details

ii)Blockchain Relevance Evaluation Framework:

We will perform the fitness assessment in this section. Let us go through the questions listed in the assessment. These are not the final set of criteria but a good set of primary criteria to be considered.

1. **Does use case involve sharing assets/valuables/data between multiple participants?**
   Yes. Multiple participants who are not confined to a single organization are involved. The fish is the asset that is shared between various participants.

2. **Is there impact due to sharing/hiding information between participants?**
   Yes, intermediaries are involved. The fish does not reach the restaurant or the supermarkets directly. They are passed through intermediaries. In this example, the intermediaries can also be accommodated in the blockchain. It should be noted that not all blockchain use cases focus on eliminating intermediaries. Some of the use cases may still involve intermediaries, but the reliance on intermediaries is reduced.

3. **Does the solution require shared write access?**
   Yes. Fish is the asset that moves through the supply chain, that is, the ownership of the fish is changed as it moves through the supply chain.

4. Can the solution work without delete?

Yes. As of today, the tuna supply chain does not mandate delete of data. Also, delete of change of hands will not help build the desired transparency.

5. Is it required to store large amount of non-transactional data?

No. The properties of fish can be generally digitized and stored on the blockchain. Any related documents shall be stored on a central database by the needed parties outside the blockchain.

6.Is it required that only a very small group or an entity needs all the control?

Yes. The parties participating in the supply chain can control the supply chain functionality.

7. Does use case involve high-performance (millisecond) transactions?

No. This process spans through various participants over a period of few days to few weeks and does not demand any high-performance transactions.

Now, we know that the assessment revealed that it is a very good use case for blockchain.

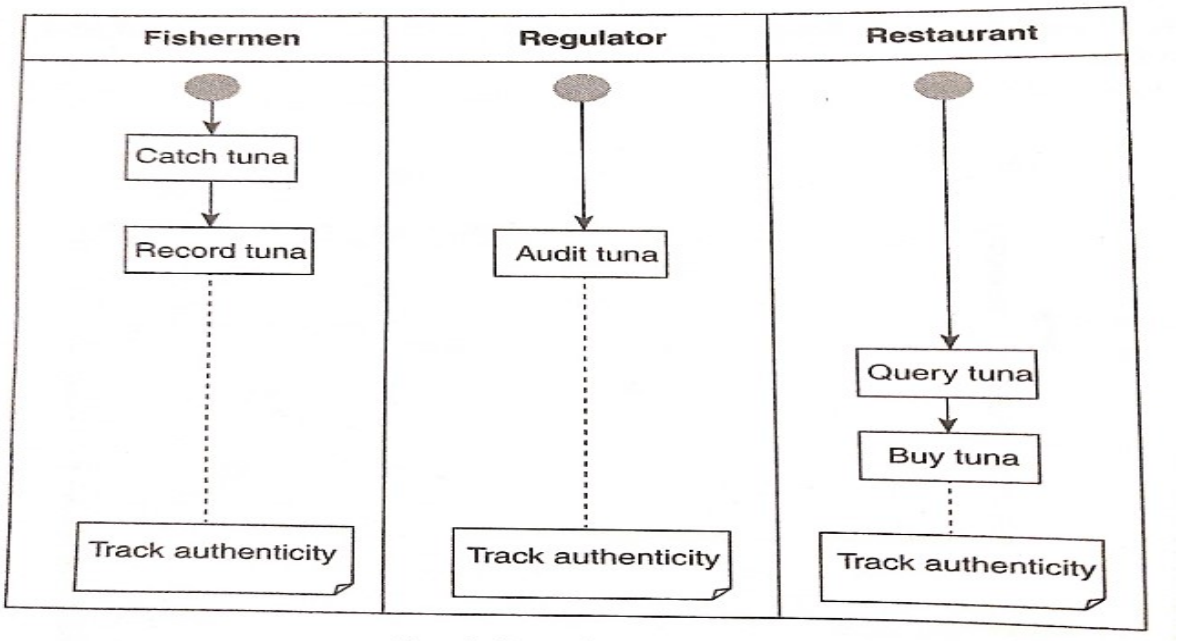
iii)Actors, Roles and Permissions:

a) Producer (Fishermen)
b) Supplier
c) Fish-Processing Unit
d)Transporter
e) Distributor
f) Retailer
g) Restaurant
h) Supermarket
i) Restaurant Customers
j) Regulator

Now we will capture the various activities that shall be performed by each of the actors. For the sake of simplicity, only free actors are chosen for the subsequence section-fishermen, regulator, and restaurant.

However, the solution approach and concepts can be extended to more number of actors as well. We will be utilizing Unified modeling language (UML) diagrams to capture these details.

iv)Activity Diagram:



Activity diagram.

v)Sequence Diagram:

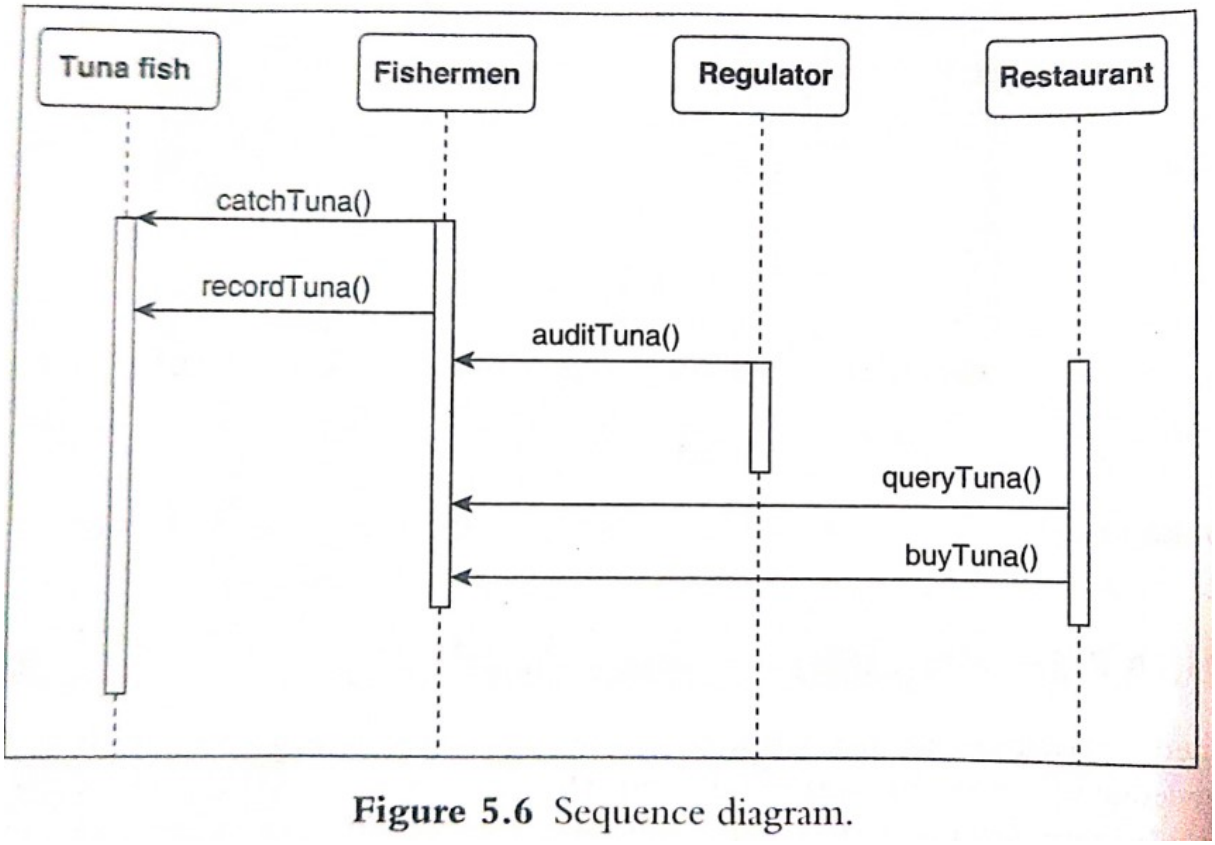


Figure 5.6 Sequence diagram.

Vi)State Diagram

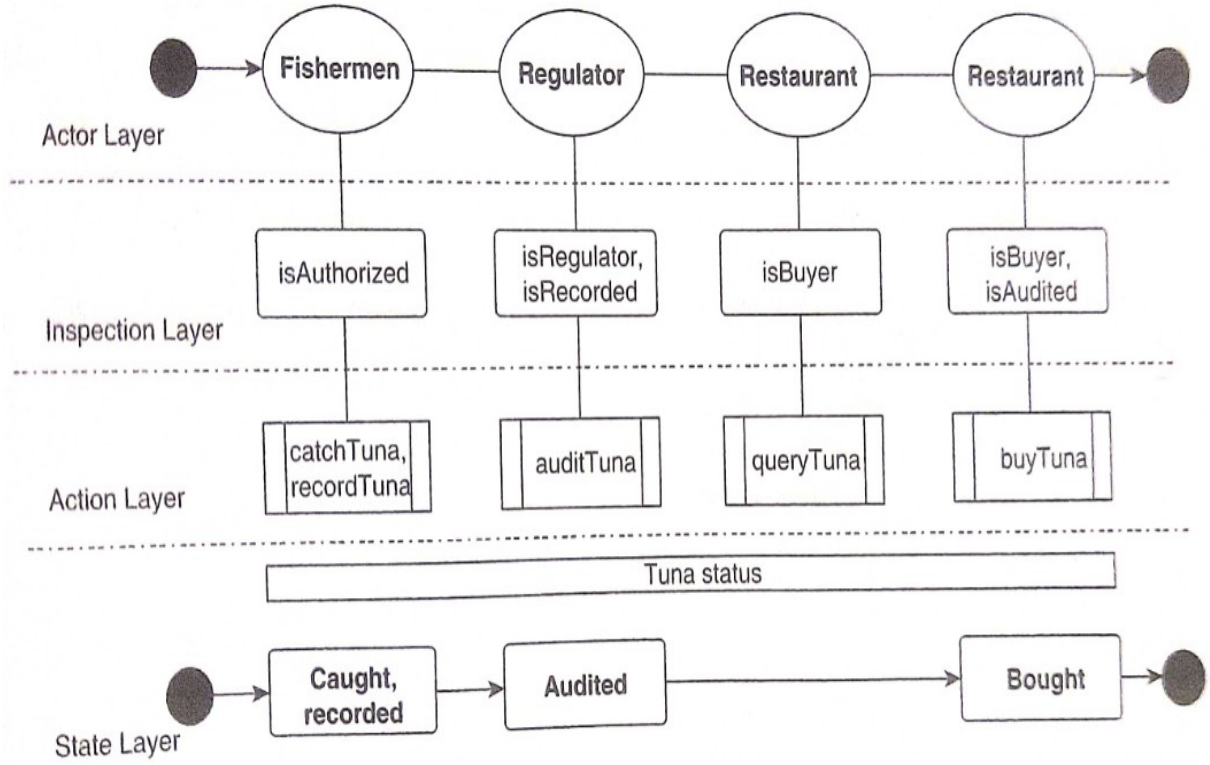


Figure 5.7 State diagram.

For further understanding, the **state diagram** is split into multiple layers. The bottom layer of Figure 5.7 marked as State Layer shows hoy the state of tuna fish changes during its journey. This gets recorded on blockchain via smart contracts.

The layer above this is represented as **Action Layer**, which shows the corresponding action that results in that particular state change.

This is accompanied by another layer known as **Inspection Layer**, which performs some simple checks to ensure that an actor is authorized to perform particular action and the current state of tuna is valid to perform that action.

For example when a regulator wants to carry **AuditTuna** activity, the following checks occur:

1. **isRegulator:** This will check whether the user is a Regulator.
2. **isRecorded:** This will ensure that the current status is "Recorded", which means that the metadata related to tuna fish is recorded on blockchain.

The topmost layer of the state diagram represented in Figure 5.7 as **Actor Layer**, shows different actors and the sequence in which they perform activities.

Vii)Class Diagram:-

On the basis of the specifications in Action layer and Inspection layer, a **class diagram** can be created (Figure 5.8).

The das diagram in Figure 5.8 gets translated into smart contracts and functions. For development VS Cole IDE with Solidity extension would be used.

For the sake of modularity and simplicity, the Solidity contracts and functions have been categorized as follows:

1. **Helper:** This deals with helper functions related to access control. For example, isRegulator, isAudited, etc.
2. **Supply Chain**: This deals with the actions and state changes performed within the application
3. **Admin:** This implements some smart contract control and maintenance functions like ownership transfer, kill, etc.
4. **Gateway:** This is the contract exposed to DApp and it inherits Supply Chain and Admin contracts.

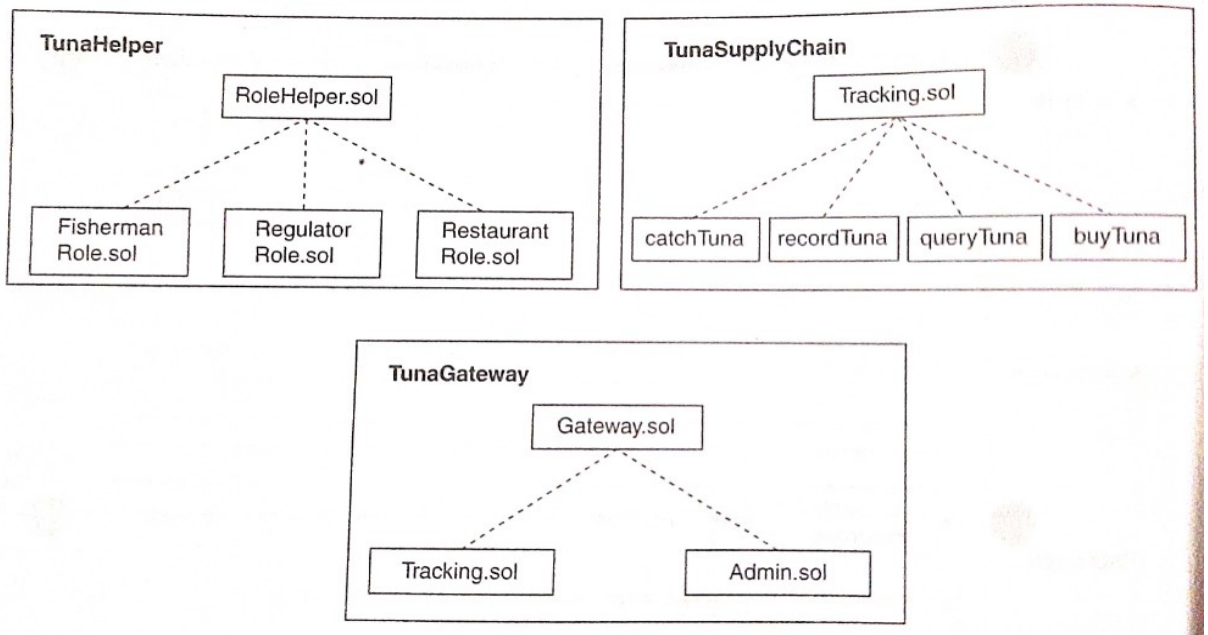
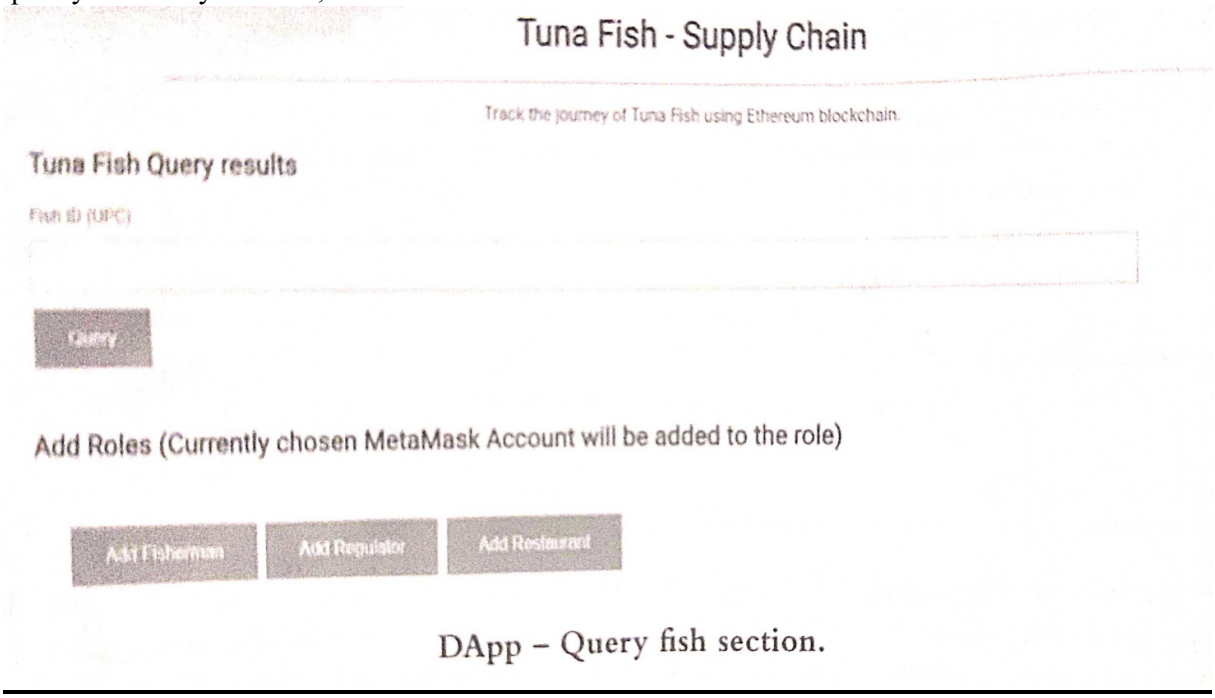


Figure 5.8 Class diagram.

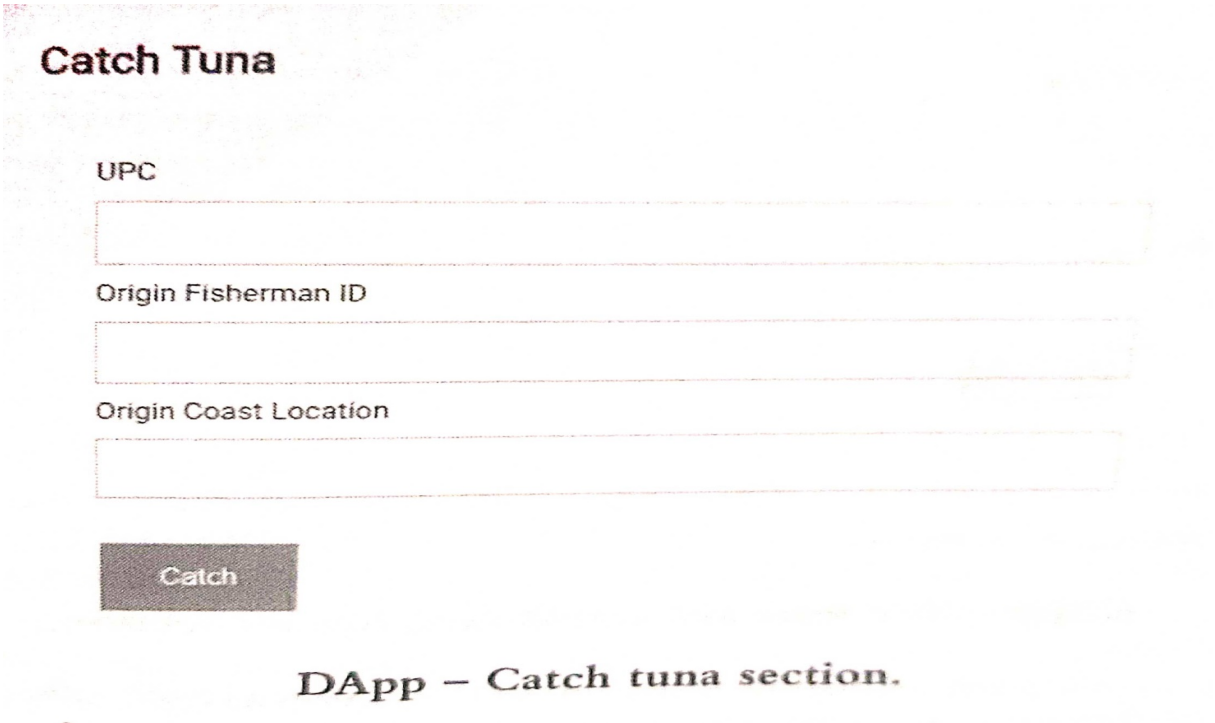
viii)Output Screen Shots:

During Execution, a simple **QR code scan** or **Fish ID** should reveal the story of tuna fish to its consumers. The data collected from RFID tracking and IoT devices can be inserted into the blockchain.

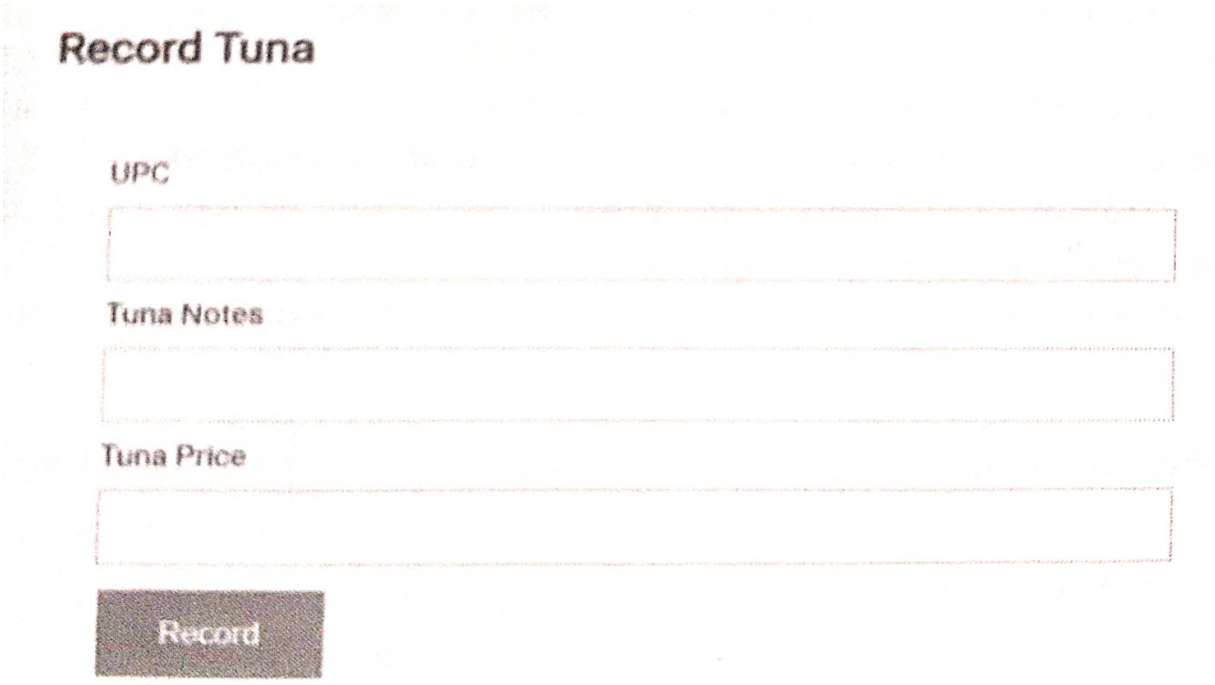
This gives details on when and where the fish was caught processing temperatures, result of quality checks by auditors, etc



DApp – Query fish section.

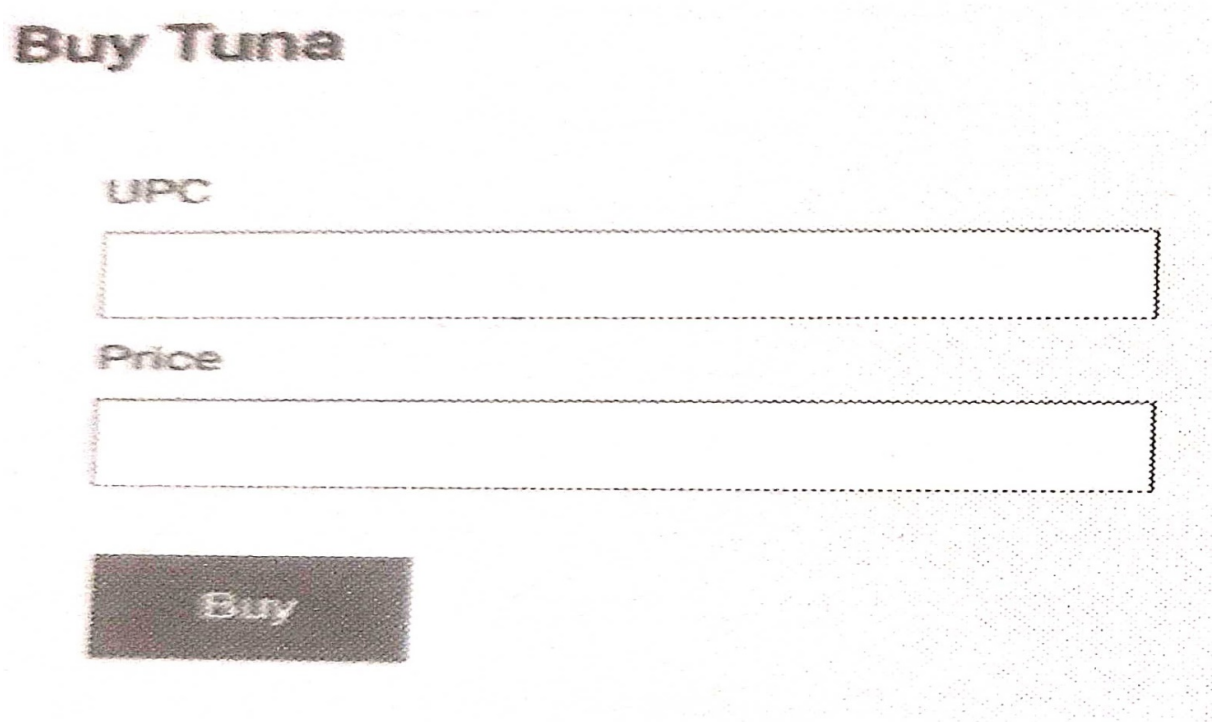


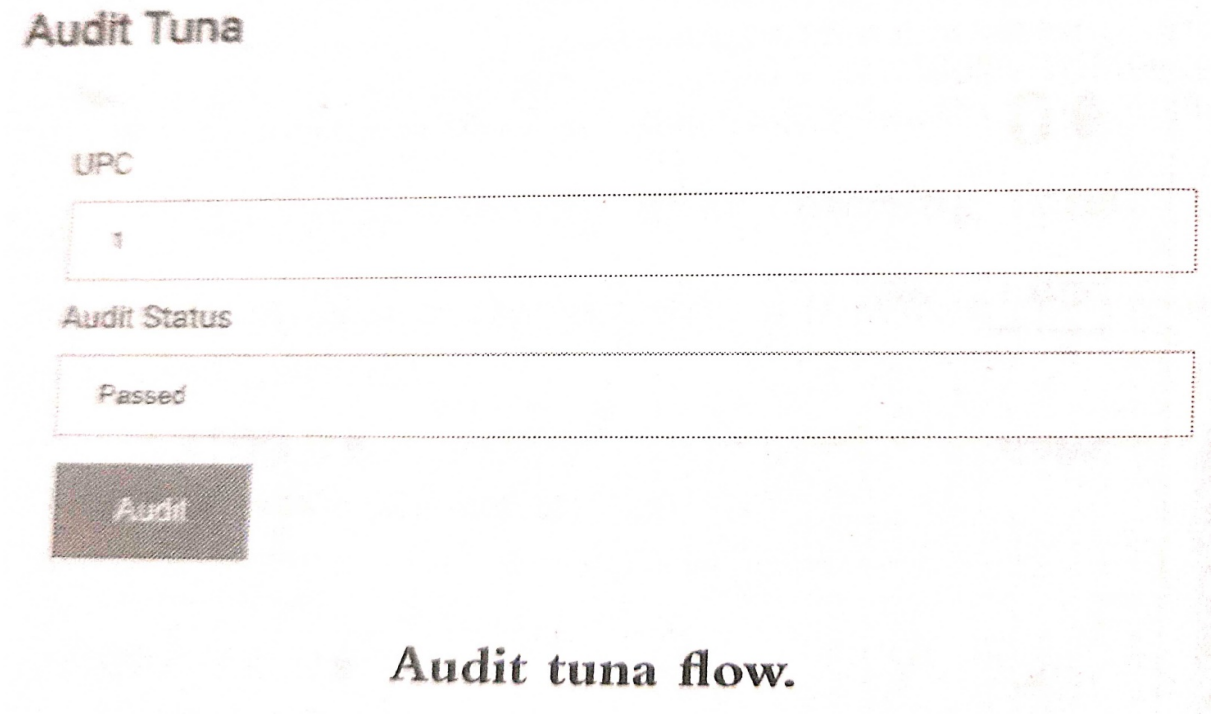
DApp – Catch tuna section.



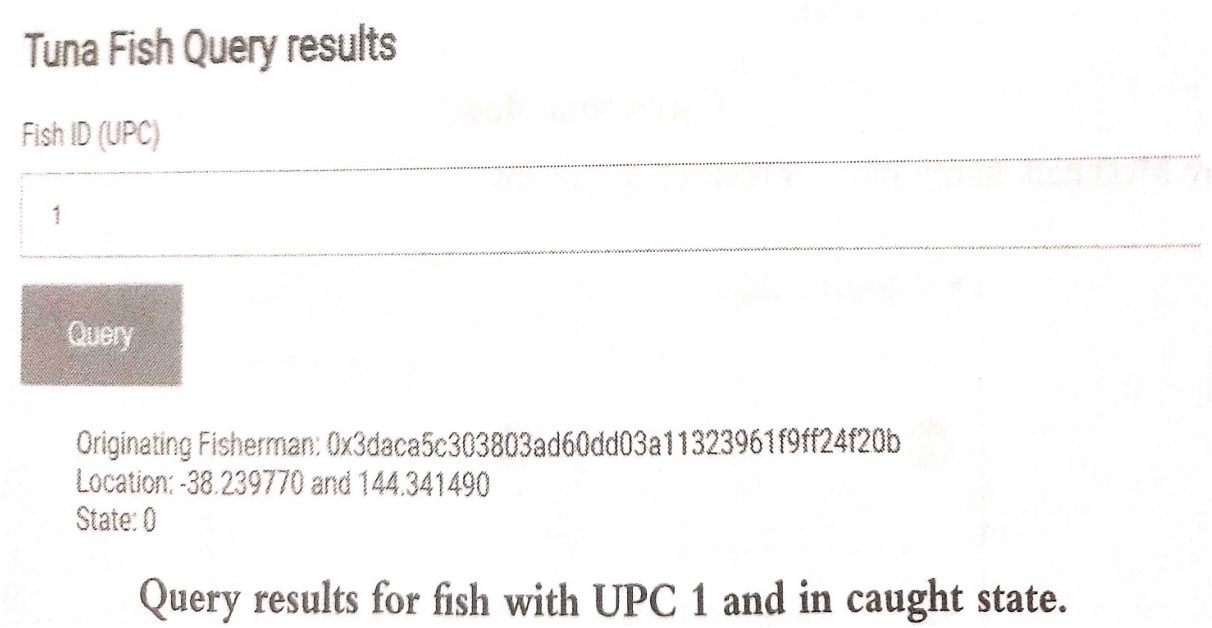
DApp – Record tuna section.

DApp – Audit tuna section.





Audit tuna flow.



Query results for fish with UPC 1 and in caught state.

Openzeppelin Concepts

Introduction:-

- OpenZeppelin is a library of secure and tested smart contracts for Ethereum and other blockchain platforms.
- Developers can use these standard implementations without the need to reinvent the wheel.
- Also, these contracts can be extended as required. It can be installed by running the following command from a node console **npm install @openzeppelin/contracts**.
- OpenZeppelin contracts implement some of the common features related to access control, ERC20 and ERC721 tokens, crowdsales, and utilities.
- There are some other useful miscellaneous libraries within utilities, namely, Address and Counter.
- Address library has a collection of functions related to address data type. For example, isContract(address), toPayable(address).
- Counter library functions can be used to increment or decrement a value by 1. This will be useful to track the number of elements in a mapping data type, issue token IDs, etc.

----------********----------