JUNAID GIRKAR
60004190057
BE COMPS A2

BLOCKCHAIN TECHNOLOGIES

EXPERIMENT - 4

AIM: Develop, Deploy smart contracts using Ganache.

THEORY:

What is a smart contract?
Smart contracts are immutable programs stored on a blockchain. They automate the execution of transactions based on predetermined conditions being met, and they are widely used to execute agreements in a decentralized manner without middlemen.

Smart contracts have particular outcomes, which are governed by immutable code, so the participants in the contract can be confident in the contract execution. No third-party involvement, no time lost — agreements are executed immediately when the conditions are met.

Smart contracts can be deployed on the blockchain for use. Ethereum supports smart contracts written in the Solidity programming language.

Benefits of Smart Contracts:

Accuracy, Speed, and Efficiency

- The contract is immediately executed when a condition is met.
- Because smart contracts are digital and automated, there is no paperwork to deal with, and
- No time was spent correcting errors that can occur when filling out documentation by hand.

Trust and Transparency

- There's no need to worry about information being tampered with for personal gain because there's no third party engaged and
- Encrypted transaction logs are exchanged among participants.

Security

- Because blockchain transaction records are encrypted, they are extremely difficult to hack.
- Furthermore, because each entry on a distributed ledger is linked to the entries before and after it, hackers would have to change the entire chain to change a single record.

Savings

- Smart contracts eliminate the need for intermediaries to conduct transactions, as well as the time delays and fees that come with them.

How Do Smart Contracts Work?

A smart contract is a sort of program that encodes business logic and operates on a dedicated virtual machine embedded in a blockchain or other distributed ledger.

Step 1: Business teams collaborate with developers to define their criteria for the smart contract's desired behavior in response to certain events or circumstances.

Step 2: Conditions such as payment authorization, shipment receipt, or a utility meter reading threshold are examples of simple events.

Step 3: More complex operations, such as determining the value of a derivative financial instrument, or automatically releasing an insurance payment, might be encoded using more sophisticated logic.

Step 4: The developers then use a smart contract writing platform to create and test the logic. After the application is written, it is sent to a separate team for security testing.

Step 5: An internal expert or a company that specializes in vetting smart contract security could be used.

Step 6: The contract is then deployed on an existing blockchain or other distributed ledger infrastructure once it has been authorized.

Step 7: The smart contract is configured to listen for event updates from an "oracle," which is effectively a cryptographically secure streaming data source, once it has been deployed.

Step 8: Once it obtains the necessary combination of events from one or more oracles, the smart contract executes.

Ganache is a personal blockchain for rapid Ethereum and Corda distributed application development. You can use Ganache across the entire development cycle; enabling you to develop, deploy, and test your dApps in a safe and deterministic environment.

The biggest advantages of using Ganache smart contract development would refer to the facility for developing, testing, and deploying your smart contracts and dApp projects in a deterministic and safe environment. You can access two different variants of Ganache, depending on the type of functionality you need. The Ganache UI is the desktop application that can offer support for Ethereum and Corda development tasks. On the other hand, you have the Ganache-CLI, which is the command-line tool and focuses specifically on Ethereum development. It is also important to note that both versions of Ganache are accessible on Linux, Mac, and Windows. The more robust command-line tool, ganache, is available for Ethereum development. It offers:

- console.log in Solidity
- Zero-config Mainnet and testnet forking
- Fork any Ethereum network without waiting to sync
- Ethereum JSON-RPC support
- Snapshot/revert state
- Mine blocks instantly, on demand, or at an interval
- Fast-forward time
- Impersonate any account (no private keys required!)
- Listens for JSON-RPC 2.0 requests over HTTP/WebSockets
- Programmatic use in Node.js
- Pending Transactions

CODE:

```
C:\Users\JARVIS>npm i truffle -g
```

OUTPUT:

```
npm WARN deprecated @types/keyv@4.2.0: This is a stub types definition.
keyv provides its own type definitions, so you do not need this
installed.
```

```
npm WARN deprecated mkdirp-promise@5.0.1: This package is broken and no
longer maintained. 'mkdirp' itself supports promises now, please switch
to that.
npm WARN deprecated request@2.88.2: request has been deprecated, see
https://github.com/request/request/issues/3142
npm WARN deprecated multibase@0.6.1: This module has been superseded by
the multiformats module
npm WARN deprecated har-validator@5.1.5: this library is no longer
supported
npm WARN deprecated multicodec@0.5.7: This module has been superseded by
the multiformats module
npm WARN deprecated uuid@3.3.2: Please upgrade  to version 7 or higher.
Older versions may use Math.random() in certain circumstances, which is
known to be problematic.  See https://v8.dev/blog/math-random for
details.
npm WARN deprecated uuid@3.4.0: Please upgrade  to version 7 or higher.
Older versions may use Math.random() in certain circumstances, which is
known to be problematic.  See https://v8.dev/blog/math-random for
details.
npm WARN deprecated multibase@0.7.0: This module has been superseded by
the multiformats module
npm WARN deprecated uuid@2.0.1: Please upgrade  to version 7 or higher.
Older versions may use Math.random() in certain circumstances, which is
known to be problematic.  See https://v8.dev/blog/math-random for
details.
npm WARN deprecated multicodec@1.0.4: This module has been superseded by
the multiformats module
npm WARN deprecated cids@0.7.5: This module has been superseded by the
multiformats module

changed 780 packages, and audited 811 packages in 4m

110 packages are looking for funding
  run `npm fund` for details

9 moderate severity vulnerabilities

To address issues that do not require attention, run:
  npm audit fix

Some issues need review, and may require choosing
a different dependency.

Run `npm audit` for details.
```

CODE:

```
C:\Users\JARVIS>truffle init
```

OUTPUT:

```
Starting init...
================

> Copying project files to C:\Users\JARVIS

Init successful, sweet!

Try our scaffold commands to get started:
  $ truffle create contract YourContractName # scaffold a contract
  $ truffle create test YourTestName         # scaffold a test

http://trufflesuite.com/docs
```

CODE:

```
C:\Users\JARVIS>truffle migrate
```

OUTPUT:

```
Compiling your contracts...
===========================
> Everything is up to date, there is nothing to compile.
Network up to date.sion list from solc-bin. Attempt #1
√ Downloading compiler. Attempt #1.
C:\Users\JARVIS>
C:\Users\JARVIS>
C:\Users\JARVIS>
C:\Users\JARVIS>
C:\Users\JARVIS>truffle migrate

Compiling your contracts...
===========================
> Compiling .\contracts\TruffleTutorial.sol
> Compilation warnings encountered:

    Warning: SPDX license identifier not provided in source file. Before
```

```
publishing, consider adding a comment containing
"SPDX-License-Identifier: <SPDX-License>" to each source file. Use
"SPDX-License-Identifier: UNLICENSED" for non-open-source code. Please
see https://spdx.org for more information.
--> project:/contracts/TruffleTutorial.sol

,Warning: Visibility for constructor is ignored. If you want the
contract to be non-deployable, making it "abstract" is sufficient.
 --> project:/contracts/TruffleTutorial.sol:8:3:
  |
8 |   constructor() public {
  |   ^ (Relevant source part starts here and spans across multiple
lines).


> Artifacts written to C:\Users\JARVIS\build\contracts
> Compiled successfully using:
   - solc: 0.8.17+commit.8df45f5f.Emscripten.clang


Starting migrations...
======================
> Network name:    'development'
> Network id:      5777
> Block gas limit: 6721975 (0x6691b7)


2_TruffleTutorial_migration.js
==============================

   Deploying 'TruffleTutorial'
   ---------------------------
   > transaction hash:
0xf71dc60cf73b35749a0b6289fcdb42417d952bc9a3c87b4f19741ed120814035
   > Blocks: 0            Seconds: 0
   > contract address:    0x2D860C0c839F0a17d65E74c4101121efF3973367
   > block number:        1
   > block timestamp:     1666248335
   > account:             0xbC84b3F616E6d046e26F83496E13Fac2aF951280
   > balance:             99.98793516
   > gas used:            603242 (0x9346a)
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.01206484 ETH
```

```
   > Saving artifacts
   -------------------------------------
   > Total cost:          0.01206484 ETH

Summary
=======
> Total deployments:   1
> Final cost:          0.01206484 ETH
```

CODE:

```
C:\Users\JARVIS>
truffle(development)> const truffleTutorial = await
TruffleTutorial.deployed()
```

OUTPUT:

```
Undefined
```

CODE:

```
truffle(development)> const address = await truffleTutorial.address
```

OUTPUT:

```
Undefined
```

CODE:

```
truffle(development)> address
```

OUTPUT:

```
'0x2D860C0c839F0a17d65E74c4101121efF3973367'
```

CODE:

```
truffle(development)> const message = await truffleTutorial.message()
```

OUTPUT:

```
Undefined
```

CODE:

```
truffle(development)> message
```

OUTPUT:

```
'Hello World!'
```

CODE:

```
truffle(development)> await truffleTutorial.setMessage('Hi there!')
```

OUTPUT:

```
{
  tx:
'0x42b708884b04bf885c9c24d24b5f633904c0d937cc97352f653e030b248aff79',
  receipt: {
    transactionHash:
'0x42b708884b04bf885c9c24d24b5f633904c0d937cc97352f653e030b248aff79',
    transactionIndex: 0,
    blockHash:
'0xdc65cca3bbcf3f08d9dfe8fb29de688215e4ddc600b05e85dee9f691275d9782',
    blockNumber: 2,
    from: '0xbc84b3f616e6d046e26f83496e13fac2af951280',
    to: '0x2d860c0c839f0a17d65e74c4101121eff3973367',
    gasUsed: 33199,
    cumulativeGasUsed: 33199,
    contractAddress: null,
    logs: [],
    status: true,
    logsBloom:
'0x00000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000
00000000000',
    rawLogs: []
  },
  logs: []
}
```

CODE:

```
truffle(development)> await truffleTutorial.message()
```

OUTPUT:

```
'Hi there!'
```

CODE:

```
truffle(development)> web3.eth.sendTransaction({to:accounts[1],
from:accounts[0], value: web3.utils.toWei('1')})
```

OUTPUT:

```
{
  transactionHash:
'0x518ce4bc421a79863af4f3adc1b88eb5cc39e4847370ea6a6276df31a278d9c9',
  transactionIndex: 0,
  blockHash:
'0xac69154ad3aba3ef6002240e7982cfb185e784576e4033544b829814bdc78ca9',
  blockNumber: 3,
  from: '0xbc84b3f616e6d046e26f83496e13fac2af951280',
  to: '0xed3d6b8b51532df0348de16395eb7b69a57ba11a',
  gasUsed: 21000,
  cumulativeGasUsed: 21000,
  contractAddress: null,
  logs: [],
  status: true,
  logsBloom:
'0x00000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000
00000000000'
}
```

CODE:

```
truffle(development)> web3.eth.sendTransaction({to:accounts[1],
from:accounts[0], value: web3.utils.toWei('1')})
```

OUTPUT:

```
{
  transactionHash:
'0x6cd103163446ca10dd7ac1dc6a36bc5580d21e4fefc30b0c84e8e313e650882d',
  transactionIndex: 0,
  blockHash:
'0x2a7e3f941919f3813231 83df7083e14a03939af96377e6c30de03bb1c2c5c50e',
  blockNumber: 4,
  from: '0xbc84b3f616e6d046e26f83496e13fac2af951280',
  to: '0xed3d6b8b51532df0348de16395eb7b69a57ba11a',
  gasUsed: 21000,
  cumulativeGasUsed: 21000,
  contractAddress: null,
  logs: [],
  status: true,
  logsBloom:
'0x00000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000
00000000000'
}
```

CODE:

```
truffle(development)> web3.eth.sendTransaction({to:accounts[1],
from:accounts[0], value: web3.utils.toWei('1')})
```

OUTPUT:

```
{
  transactionHash:
'0x8a791ec519e3e8bfca0a1ee5d943f2027f085f8a9def68e444339c28fcecd63a',
  transactionIndex: 0,
  blockHash:
'0x4a2f1d66fd3db3b09f5b667b936bcb6ace807fce6cb56d2cd3539b565b3aabf0',
  blockNumber: 5,
  from: '0xbc84b3f616e6d046e26f83496e13fac2af951280',
  to: '0xed3d6b8b51532df0348de16395eb7b69a57ba11a',
  gasUsed: 21000,
  cumulativeGasUsed: 21000,
  contractAddress: null,
```

```
  logs: [],
  status: true,
  logsBloom:
'0x000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
00000000000'
}
```

CODE:

```
truffle(development)> web3.eth.sendTransaction({to:accounts[1],
from:accounts[0], value: web3.utils.toWei('1')})
```

OUTPUT:

```
{
  transactionHash:
'0x16a70ee9432068d4f8d8ceceb044c9df8a5e7ce31a4e048219d96f318aea8940',
  transactionIndex: 0,
  blockHash:
'0x6497473c2019b902be07e247ce90980639c2aab79244f41fc24e910d50d84023',
  blockNumber: 6,
  from: '0xbc84b3f616e6d046e26f83496e13fac2af951280',
  to: '0xed3d6b8b51532df0348de16395eb7b69a57ba11a',
  gasUsed: 21000,
  cumulativeGasUsed: 21000,
  contractAddress: null,
  logs: [],
  status: true,
  logsBloom:
'0x000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
00000000000'
}
```

CODE:

```
C:\Users\JARVIS>npm i chai chai-as-promised
```

OUTPUT:

```
added 9 packages, and audited 10 packages in 2m

found 0 vulnerabilities
```

CODE:

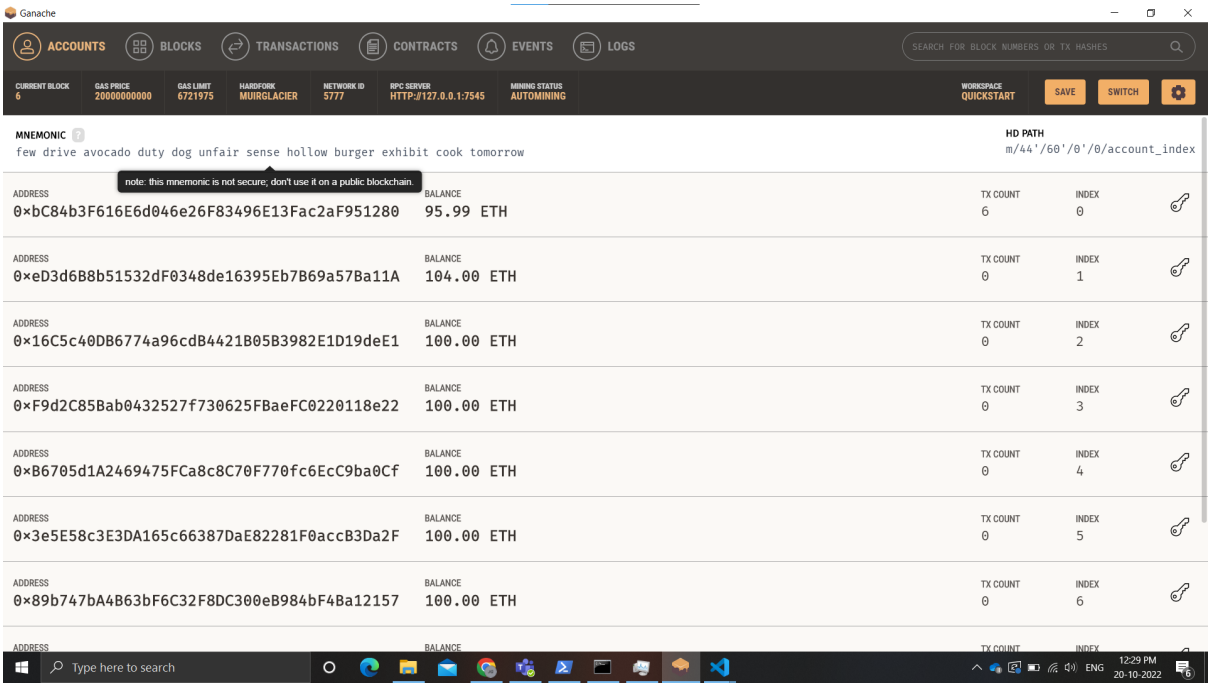
```
C:\Users\JARVIS>truffle test
```

OUTPUT:

```
Using network 'development'.


Compiling your contracts...
===========================
> Compiling .\contracts\TruffleTutorial.sol
> Compilation warnings encountered:

    Warning: SPDX license identifier not provided in source file. Before
publishing, consider adding a comment containing
"SPDX-License-Identifier: <SPDX-License>" to each source file. Use
"SPDX-License-Identifier: UNLICENSED" for non-open-source code. Please
see https://spdx.org for more information.
--> project:/contracts/TruffleTutorial.sol

,Warning: Visibility for constructor is ignored. If you want the
contract to be non-deployable, making it "abstract" is sufficient.
 --> project:/contracts/TruffleTutorial.sol:8:3:
  |
8 |   constructor() public {
  |   ^ (Relevant source part starts here and spans across multiple
lines).


> Artifacts written to
C:\Users\JARVIS\AppData\Local\Temp\test--11892-nMJ4r84E14OD
> Compiled successfully using:
   - solc: 0.8.17+commit.8df45f5f.Emscripten.clang
```

```
Contract: TruffleTutorial
  deployment
    √ deploys successfully
    √ has a message (200ms)
  message
    √ contract owner sets a message (998ms)
    √ address that is not the owner fails to set a message (2785ms)


4 passing (5s)
```

Wallet after doing transactions:



CONCLUSION:

In this experiment, we learnt about Smart Contracts and deployed them on local network using Ganache. We also performed transactions using the Ganache accounts.