UNIVERSITY OF MUMBAI

DEPARTMENT OF COMPUTER SCIENCE

मुंबई विद्यापीठ

University of Mumbai

Re-accredited with A++ Grade

(CGPA 3.65) by NAAC (3rd Cycle 2021)

M.Sc. Computer Science – Semester III

Web 3 Technologies

JOURNAL

2024-2025

Seat No. ___________________

मुंबई विद्यापीठ
University of Mumbai
Re-accredited with A++ Grade
(CGPA 3.65) by NAAC (3rd Cycle 2021)

UNIVERSITY OF MUMBAI

DEPARTMENT OF COMPUTER SCIENCE

CERTIFICATE

This is to certify that the work entered in this journal was done in the University Department of Computer Science laboratory by Mr./Ms.______________________________ Seat No. ______________ for the course of M.Sc. Computer Science - Semester III (NEP 2020) during the academic year 2024- 2025 in a satisfactory manner.

____

Subject In-charge

____

Head of Department

____

External Examiner

INDEX

PRACTICAL-01

Aim:- Install and understand Docker container, Node.js, Java and Hyperledger Fabric, Ethereum and perform necessary software installation on local machine/create instance on Cloud to run.

Procedure:

1. Install the prerequisite software required for Hyperledger Fabric based on your System following the instructions here : https://hyperledger-fabric.readthedocs.io/en/release-2.5/prereqs.html
2. Install Hyperledger Fabric and Fabric samples following instructions on this page : https://hyperledger-fabric.readthedocs.io/en/release-2.5/install.html
3. Install NodeJS using the instruction from : http://nodejs.org/en/download
4. Download and Install Java from : https://openjdk.org/install/
5. For Ethereum based projects, use : https://remix.ethereum.org/

Outcome:

A local development environment set up for blockchain-based application development using hyperledger fabric and a test network for deployment and testing.

Practical No. 2

Aim - Create and deploy a block chain network using Hyperledger Fabric SDK

Directory Structure:

```
my-fabric-network/          # base directory
├── crypto-config/          # Cryptogen output
├── docker/
│   ├── docker-compose.yaml   # Network definition
│   └── base/               # Peer/Orderer configurations
├── configtx.yaml           # Channel configuration
├── crypto-config.yaml      # Cryptographic material configuration
├── chaincode/
│   └── mychaincode/        # Custom chaincode
├── bin/
│   ├── configtxgen
│   ├── configtxlator
│   ├── cryptogen
│   ├── discover
│   ├── fabric-ca-client
│   ├── fabric-ca-server
│   ├── ledgerutil
│   ├── orderer
│   ├── osnadmin
│   └── peer
└── application/            # Client Application
```

File Contents:

Cryptoconfig.yaml:

```
OrdererOrgs:
  - Name: Orderer
    Domain: example.com
    Specs:
      - Hostname: orderer

PeerOrgs:
  - Name: Org1
    Domain: org1.example.com
    EnableNodeOUs: true
    Template:
      Count: 1
    Users:
      Count: 1
```

configtx.yaml

```
Organizations:
  - &OrdererOrg
    Name: OrdererOrg
    ID: OrdererMSP
    MSPDir: crypto-config/ordererOrganizations/example.com/msp
    Policies:
      Readers:
        Type: Signature
        Rule: "OR('OrdererMSP.member')"
      Writers:
        Type: Signature
        Rule: "OR('OrdererMSP.member')"
      Admins:
        Type: Signature
        Rule: "OR('OrdererMSP.admin')"
      BlockValidation:
        Type: ImplicitMeta
        Rule: "ANY Writers"

  - &Org1
    Name: Org1MSP
    ID: Org1MSP
    MSPDir: crypto-config/peerOrganizations/org1.example.com/msp
    Policies:
      Readers:
        Type: Signature
        Rule: "OR('Org1MSP.admin', 'Org1MSP.peer', 'Org1MSP.client')"
      Writers:
        Type: Signature
        Rule: "OR('Org1MSP.admin', 'Org1MSP.client')"
      Admins:
        Type: Signature
        Rule: "OR('Org1MSP.admin')"
      Endorsement:
        Type: Signature
        Rule: "OR('Org1MSP.peer')"

Capabilities:
  Channel: &ChannelCapabilities
    V2_0: true
  Orderer: &OrdererCapabilities
    V2_0: true
  Application: &ApplicationCapabilities
    V2_0: true

Application: &ApplicationDefaults
  Organizations:
  Policies:
    Readers:
      Type: ImplicitMeta
      Rule: "ANY Readers"
    Writers:
      Type: ImplicitMeta
      Rule: "ANY Writers"
    Admins:
      Type: ImplicitMeta
      Rule: "MAJORITY Admins"
    LifecycleEndorsement:
      Type: ImplicitMeta
      Rule: "MAJORITY Endorsement"
    Endorsement:
      Type: ImplicitMeta
      Rule: "MAJORITY Endorsement"

Orderer: &OrdererDefaults
  OrdererType: etcdraft
  Addresses:
    - orderer.example.com:7050
  BatchTimeout: 2s
  BatchSize:
    MaxMessageCount: 10
    AbsoluteMaxBytes: 99 MB
    PreferredMaxBytes: 512 KB
  EtcdRaft:
    Consenters:
      - Host: orderer.example.com
        Port: 7050
      ClientTLSCert: crypto-config/ordererOrganizations/example.com/orderers/orderer.example.com/tls/server.crt
      ServerTLSCert: crypto-config/ordererOrganizations/example.com/orderers/orderer.example.com/tls/server.crt
  Organizations:
  Policies:
    Readers:
      Type: ImplicitMeta
      Rule: "ANY Readers"
    Writers:
      Type: ImplicitMeta
      Rule: "ANY Writers"
    Admins:
      Type: ImplicitMeta
      Rule: "MAJORITY Admins"
    BlockValidation:
      Type: ImplicitMeta
      Rule: "ANY Writers"

Channel: &ChannelDefaults
  Policies:
    Readers:
      Type: ImplicitMeta
      Rule: "ANY Readers"
    Writers:
      Type: ImplicitMeta
      Rule: "ANY Writers"
    Admins:
      Type: ImplicitMeta
      Rule: "MAJORITY Admins"

Profiles:
  SampleSingleMSPChannel:
    <<: *ChannelDefaults
    Consortium: SampleConsortium
    Application:
      <<: *ApplicationDefaults
      Organizations:
        - *Org1
    Orderer:
      <<: *OrdererDefaults
      Organizations:
        - *OrdererOrg
```

docker-compose.yaml:

```
networks:
  fabric:

services:
  orderer.example.com:
    container_name: orderer
    image: hyperledger/fabric-orderer:2.5
    environment:
      - ORDERER_GENERAL_LISTENPORT=7050
      - ORDERER_GENERAL_LOCALMSPID=OrdererMSP
    volumes:
      - ../crypto-config/ordererOrganizations/example.com/orderers/orderer.example.com:/etc/hyperledger/fabric
    ports:
      - 7050:7050
    networks:
      - fabric

  peer0.org1.example.com:
    container_name: peer0_org1
    image: hyperledger/fabric-peer:2.5
    environment:
      - CORE_PEER_ID=peer0.org1.example.com
      - CORE_PEER_ADDRESS=peer0.org1.example.com:7051
      - CORE_PEER_LOCALMSPID=Org1MSP
```

```
volumes:
  - ../crypto-config/peerOrganizations/org1.example.com/peers/peer0.org1.example.com:/etc/hyperledger/fabric
ports:
  - 7051:7051
depends_on:
  - orderer.example.com
networks:
  - fabric
```

Steps:

1. Open a terminal and navigate to the base directory using "*cd*" command.
2. Generate certificates using the command "*./bin/cryptogen generate --config=crypto-config.yaml --output=crypto-config*".
3. Generate Gensis block using the command "*./bin/configtxgen -profile SampleSingleMSPChannel -outputBlock genesis.block -channelID system-channel*".
4. Generate channel transaction using the command "./bin/configtxgen -profile SampleSingleMSPChannel -outputCreateChannelTx mychannel.tx -channelID mychannel".
5. Start the network using the command "docker compose -f docker/docker-compose.yaml up -d".
6. Create a channel using the command "*docker exec -it peer0_org1 bash export CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/fabric/msp peer channel create -o orderer.example.com:7050 -c mychannel -f /etc/hyperledger/fabric/channel.tx*".
7. Query the channel policies using the command "*./bin/peer channel getinfo –c mychannel*".
8. Shut the network using the command "*docker compose -f docker/docker-compose.yaml down*" and clean up using the command, "*docker volume prune*".

Outcome:

A private local blockchain deployed using the Hyperledger Fabric Framework.

Practical No.3

Aim: Interact with a blockchain Network. Execute transactions and requests against a blockchain network.

Program:

[Smart Contract]

```
pragma solidity ^0.8.0;
contract SimpleStorage {
   uint256 public storedData;
   // Event to log changes
   event ValueChanged(uint256 newValue);
   // Function to store data
   function set(uint256 _value) public {
      storedData = _value;
      emit ValueChanged(storedData);
   }
   // Function to retrieve data
   function get() public view returns (uint256) {
      return storedData;
   }
}
```

Procedure:

1. Open a browser and enter “remix.ethereum.org” in the address bar and visit the Remix IDE.
2. Click on the File Explorer button to see the file pane.
3. Create a new file in the “contracts” directory by the name of “simple.sol”.
4. Enter the Program into the file and compile the code using “CTRL+S” shortcut or the green play button on the ribbon.
5. Navigate to “Deploy and run Transactions” pane to deploy the smart contract to a selected test network.
6. Select one of the available accounts or create a new one and deploy the smart contract
7. Deployed Contracts are available to utilise in the same pane under the “Deployed Transactions” Section.
8. Use a deployed smart contract to run transaction, by selecting it and filling up the required details, make sure the user selected to execute the smart contract has enough gas, otherwise the contract will not be processed.

Outcome:

A simple smart contract to interact with a blockchain test network and understand its policies.

Practical no. 4

Aim: Deploy an asset transfer app using block chain.

Due to configuration concerns, this lab is to be performed using Ethereum.

Program:

```
pragma solidity ^0.8.0;
contract AssetTransfer {
   struct Asset {
      uint256 id;
      string name;
      address owner;
   }
   uint256 private nextAssetId;
   mapping(uint256 => Asset) public assets;
   event AssetCreated(uint256 id, string name, address owner);
   event AssetTransferred(uint256 id, address from, address to);
   // Create a new asset
   function createAsset(string memory _name) public {
      uint256 assetId = nextAssetId;
      assets[assetId] = Asset(assetId, _name, msg.sender);
      emit AssetCreated(assetId, _name, msg.sender);
      nextAssetId++;
   }
   // Transfer an asset to a new owner
   function transferAsset(uint256 _assetId, address _newOwner) public {
      require(assets[_assetId].owner == msg.sender, "You are not the owner");
      address previousOwner = assets[_assetId].owner;
      assets[_assetId].owner = _newOwner;
      emit AssetTransferred(_assetId, previousOwner, _newOwner);
   }
   // Get details of an asset
   function getAsset(uint256 _assetId) public view returns (uint256, string memory, address) {
      Asset memory asset = assets[_assetId];
      return (asset.id, asset.name, asset.owner);
   }}
```

Procedure:

1. Open a browser and enter “remix.ethereum.org” in the address bar and visit the Remix IDE.
2. Click on the File Explorer button to see the file pane.
3. Create a new file in the “contracts” directory by the name of “AssetTransfer.sol”.
4. Enter the Program into the file and compile the code using “CTRL+S” shortcut or the green play button on the ribbon.
5. Navigate to “Deploy and run Transactions” pane to deploy the smart contract to a selected test network.
6. Select one of the available accounts or create a new one and deploy the smart contract
7. Deployed Contracts are available to utilise in the same pane under the “Deployed Transactions” Section.
8. Use a deployed smart contract to run transaction, by selecting it and filling up the required details, make sure the user selected to execute the smart contract has enough gas, otherwise the contract will not be processed.

Outcome: A basic blockchain based asset transfer application.

Practical no. 5

Aim: Use blockchain to track fitness club rewards.

Due to configuration concerns, this lab is to be performed using Ethereum.

Program:

```
pragma solidity ^0.8.0;

contract FitnessClubRewards {
    struct Member {
        uint256 rewards; // Total rewards earned
        bool isMember;   // Membership status
    }

    address public owner;
    mapping(address => Member) public members;

    event MemberRegistered(address indexed member);
    event RewardsEarned(address indexed member, uint256 points);
    event RewardsRedeemed(address indexed member, uint256 points);

    modifier onlyOwner() {
        require(msg.sender == owner, "Only owner can perform this action");
        _;
    }

    modifier onlyMember() {
        require(members[msg.sender].isMember, "You must be a registered member");
        _;
    }

    constructor() {
        owner = msg.sender;
    }

    // Register a new member
    function registerMember(address _member) public onlyOwner {
        require(!members[_member].isMember, "Already a member");
        members[_member] = Member(0, true);
        emit MemberRegistered(_member);
    }

    // Earn rewards for fitness activities
    function earnRewards(address _member, uint256 _points) public onlyOwner {
        require(members[_member].isMember, "Not a registered member");
        members[_member].rewards += _points;
        emit RewardsEarned(_member, _points);
    }

    // Redeem rewards
    function redeemRewards(uint256 _points) public onlyMember {
        require(members[msg.sender].rewards >= _points, "Insufficient rewards");
        members[msg.sender].rewards -= _points;
        emit RewardsRedeemed(msg.sender, _points);
    }
    // View reward balance
    function viewRewards(address _member) public view returns (uint256) {
        return members[_member].rewards;
    }
}
```

Procedure:

1. Open a browser and enter “remix.ethereum.org” in the address bar and visit the Remix IDE.
2. Click on the File Explorer button to see the file pane.
3. Create a new file in the “contracts” directory by the name of “FitnessClubRewards.sol”.
4. Enter the Program into the file and compile the code using “CTRL+S” shortcut or the green play button on the ribbon.
5. Navigate to “Deploy and run Transactions” pane to deploy the smart contract to a selected test network.
6. Select one of the available accounts or create a new one and deploy the smart contract
7. Deployed Contracts are available to utilise in the same pane under the “Deployed Transactions” Section.
8. Use a deployed smart contract to run transaction, by selecting it and filling up the required details, make sure the user selected to execute the smart contract has enough gas, otherwise the contract will not be processed.

Outcome: An ethereum blockchain based application to track reward points of members of a fitness club.

Practical no.6

Aim: Build a webapp that uses hyperledger fabric to track and trace member rewards.

Due to configuration concerns, this lab is to be performed using Ethereum.

Program:

```
pragma solidity ^0.8.0;
 contract FitnessClubRewards {
   struct Member {
      uint256 rewards; // Total rewards earned
      bool isMember;   // Membership status
   }
   address public owner;
   mapping(address => Member) public members;

   event MemberRegistered(address indexed member);
   event RewardsEarned(address indexed member, uint256 points);
   event RewardsRedeemed(address indexed member, uint256 points);

   modifier onlyOwner() {
      require(msg.sender == owner, "Only owner can perform this action");
      _;
   }
   modifier onlyMember() {
      require(members[msg.sender].isMember, "You must be a registered member");
      _;
   }
   constructor() {
      owner = msg.sender;
   }
   // Register a new member
   function registerMember(address _member) public onlyOwner {
      require(!members[_member].isMember, "Already a member");
      members[_member] = Member(0, true);
      emit MemberRegistered(_member);
   }
   // Earn rewards for fitness activities
   function earnRewards(address _member, uint256 _points) public onlyOwner {
      require(members[_member].isMember, "Not a registered member");
      members[_member].rewards += _points;
      emit RewardsEarned(_member, _points);
   }
   // Redeem rewards
   function redeemRewards(uint256 _points) public onlyMember {
      require(members[msg.sender].rewards >= _points, "Insufficient rewards");
      members[msg.sender].rewards -= _points;
      emit RewardsRedeemed(msg.sender, _points);
   }
   // View reward balance
   function viewRewards(address _member) public view returns (uint256) {
      return members[_member].rewards;
   }
}
```

Procedure:

1. Open a browser and enter “remix.ethereum.org” in the address bar and visit the Remix IDE.
2. Click on the File Explorer button to see the file pane.
3. Create a new file in the “contracts” directory by the name of “AssetTransfer.sol”.

4. Enter the Program into the file and compile the code using “CTRL+S” shortcut or the green play button on the ribbon.
5. Navigate to “Deploy and run Transactions” pane to deploy the smart contract to a selected test network.
6. Select one of the available accounts or create a new one and deploy the smart contract
7. Deployed Contracts are available to utilise in the same pane under the “Deployed Transactions” Section.
8. Use a deployed smart contract to run transaction, by selecting it and filling up the required details, make sure the user selected to execute the smart contract has enough gas, otherwise the contract will not be processed.

Outcome: A blockchain based app to track and trace member rewards.