A REPORT

ON

SPORTS FANTASY APPLICATION

BY

Jaspreet Singh 2021MT93340

AT

New Delhi

Publicis Sapient, Gurugram

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI

(April, 2023)

A REPORT

ON

SPORTS FANTASY APPLICATION

BY

Jaspreet Singh	2021MT93340	M.Tech Software Engineering

Prepared in partial fulfillment of the
WILP Dissertation

AT

Publicis Sapient
Gurugram

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to all those who have contributed to the successful completion of this report. Without their guidance, support, and encouragement, this work would not have been possible.

First and foremost, I would like to thank the Parmjeet Virdi, Director Data Analytics, Publicis Sapient for providing me with the opportunity to work on this project and for the resources and facilities made available to me throughout the course of the work.I would like to express my heartfelt thanks to Swati Chandak & Rohan Dhamija for their valuable guidance, insightful feedback, and constructive criticism, which greatly improved the quality of this report.

I would like to extend my sincere appreciation to Nagesh Tavarekere Rama Moorthy for his expert knowledge, continuous support, and motivation, which were crucial to the successful completion of this work.

I would also like to thank all the other persons who have contributed in various ways to this work, from the organization and/or outside the organization. Their contributions have been invaluable, and I am deeply grateful for their support. Finally, I would like to express my gratitude to my family and friends for their unwavering support and encouragement throughout my academic journey.

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI

(April, 2023)

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI

(RAJASTHAN)

WILP Division

Organization: ...Publicis Sapient…

Location: ...Gurugram……

Duration ....3.5 Months.....

Date of Start ...7 January 2023.....

Date of Submission ......22 April 2023……

Title of the Project: SPORTS FANTASY APPLICATION

ID No./Name of the student : 2021MT93340 - Jaspreet Singh

Supervisor: Swati Chandak, Senior Associate, Data Analytics

Additional Examiner : Rohan Dhamija, Senior Associate, Data Engineering

Faculty mentor:- Swati Chandak

Key Words: Sports Fantasy, Quiz, Blockchain, MCQ game, Application, Cricket, Football, Match Prediction, Data regression

Project Areas:

Blockchain Technology is a great way to ensure transparency and fairness in any game or software. The broad range of our project include Blockchain, Machine Learning, and Database design techniques. The current research project focus in the following application areas:

- Blockchain: To ensure transparency and fairness in reward distribution
- Machine learning: To adapt and detect new patterns
- Database design: To fetch cricket results from APIs and process them to store results in local DB.

ABSTRACT

Blockchain Technology is a great way to ensure transparency and fairness in any game or software. The broad range of our project include Blockchain, Machine Learning, and Database design techniques. The current research project focus in the following application areas:

- Blockchain: To ensure transparency and fairness in reward distribution
- Machine learning: To adapt and detect new patterns
- Database design: To fetch cricket results from APIs and process them to store results in local DB.

The objectives of my project are as follows:

- Understand the current scenario in the market with respect to the sports fantasy applications.
- Designing the smart contracts in Solidity that uses sports APIs via oracles to fetch match scores of cricket and do the processing fairly.
- Building a mobile app in Flutter for cricket prediction questions and developing the APIs for fetching results.
- Prediction of results from a Machine learning trained model that uses track record of the past will be used to assign option's participation fees. More the participation fee, more should be the probability of that option to be correct

The scope of this dissertation is to develop smart contracts and mobile app that allows users to predict cricket match outcomes in the form of simple question answers. The multiple options of question will have different participation fees based on Machine learning trained dataset.

Jaspreet Singh
Signature of the Student

Swati Chandak
Signature of the Supervisor

Name: Swati Chandak
Date: 17/04/2023
Place: Delhi

Contents

1. OVERVIEW

In this dissertation, we explore the development of a blockchain-based quiz platform that allows users to participate in quizzes and win rewards. The platform uses blockchain smart contracts and a sports API to fetch real-time data and generate quiz questions. The system also maintains an off-chain database to store user data and other relevant information.

One of the key challenges in developing a blockchain-based quiz platform is to ensure that the transactions are scalable and cost-effective. In the market, we see various sports fantasy apps that have centralized databases and there is no possible way for customers to go and validate if the winners were rigged up. Therefore, we propose using Polygon, an EVM-compatible blockchain that uses Proof of Stake (PoS) consensus. By using Polygon, we can reduce the cost of transactions and improve scalability. In this dissertation, we use Solidity to code the smart contracts and deploy them on the Polygon testnet for testing.

The smart contracts developed for the platform have several requirements that need to be fulfilled.

1. First, the system needs to be able to generate quiz questions on-chain. To achieve this, we develop a function that creates match questions on-chain, which are generated by a Python script. The question content is stored on-chain as well, and we can store both integer type questions (type 1) and MCQ type questions (type 2). The function takes several parameters, including the question ID (_qid), question content (_qcontent), participation fee (_pfee), number of options (_optionslength), question type (_type), and the name of the question creator. We concatenate the Match ID (returned by the SportsMonk API) and question number generated by progression (1, 2, 3 and so on) to create the question ID. We also allow the name of the question creator to be sent in the transaction for quick identification, and it will be null if created by scripts automatically.

2. Second, the system needs to allow users to participate in match questions on-chain using question ID and answer ID. We develop a function that enables users to participate in match questions using their public keys and other relevant information.
3. Third, the system needs to declare results automatically after the match finishes. We fetch the results in real-time using Python scripts that use the SportsMonk free APIs for proof of concept. The results are submitted to the smart contract using a function that takes several parameters, including the question ID (_qid), answer ID (_ansid), the correct answer string (_correct_answer_str), and the correct answer reason (_correct_ans_reason).
4. Fourth, the system needs to assign the winnings for each quiz question. The winnings are calculated based on the answers chosen by users while participating in the quiz. For integer type questions, the user with the closest prediction of the match answer gets all the participation fee collected. If there is more than one user with the same answer, the winnings are distributed equally. For MCQ type questions, the users with the correct MCQ option selected get the winnings. All the users who selected other options will have the participation fee collected from those users distributed equally amongst the winners.
5. Fifth, the system needs to allow anyone to fetch the total winnings using their account address for transparency. We develop a function that enables anyone to fetch the total winnings using their account address.
6. Sixth, the system needs to allow users to retrieve their winnings. When the balance of platform tokens is transferred to the smart contract, the mobile app can give FIAT winnings to users after cutting the platform fees.

In addition to the smart contracts, we also develop an off-chain database to store user data and other relevant information. We use MongoDB to store the data, which is continuously updated using Python scripts that fetch cricket and football data from SportsMonk APIs. The data is transformed and stored in the off-chain database to reduce calls on third-party APIs and make the API calls on the mobile application more effective. The off-chain database has several collections, including the Users collection and others required.

2. BLOCKCHAIN SMART CONTRACTS

Smart contracts need to be developed as a source of truth. At the same time, we should not use expensive chains as for every transaction we may need to pay as an application. So, using chains like Polygon to scale the transactions and using Proof Of Stake (PoS) consensus. Since Polygon is an EVM compatible chain, we will be coding smart contracts in Solidity and deploy on Polygon testnet for sake of testing in this dissertation.

Smart Contract Requirements:

(a) Quiz questions generation

(b) Participation from different accounts (addresses)

(c) Declare result automatically after match finishes

(d) To assign the winnings for each quiz questions

(e) Fetch the winnings of any address publicly

(f) Retrieve the winnings by users

The description of each of the requirement is described below:

(a) Quiz questions generation: The function should be able to create match questions onchain generated by the python script. We are also storing question content (_qcontent) onchain as it doesn't make much noticeable cost difference on Polygon chain. We can both integer type question, referred as type 1 and MCQ type question referred as type 2.

```
function postQuestion(
        uint256 _qid,
        string memory _qcontent,
        uint256 _pfee,
        uint8 _optionslength,
        uint256 _type,
        string memory name)
```

The _qid for the smart contract is created by concatenating Match ID (returned by SportsMonk API) and question number generated by progression (1, 2, 3 and so on)

_qid = Match ID + Question No.

The pseudo name (to secure PII) of the question creator can be sent in the transaction

for quick identification. It will be null if created by scripts automatically. The name can be saved in the mobile app and referred to as XID in the application when the user signs up.

If required, we can just keep question id, question type and participation fee to save transaction cost. We can use the IPFS hash function to store all the data.

(b) Participation from different accounts (addresses): The function should be able to allow users to participate in match questions on chain using question id and answer id.

```
function participate(
        uint256 _qid,
        uint256 ansid,
        string memory name)
```

(c) Declare result automatically after match finishes: After the cricket/football match finishes in real time, the results are fetched in Python scripts using APIs (using SportsMonk free APIs for POC).

```
function submitRightSolution(
        uint256 _qid,
        uint256 _ansid,
        string memory _correct_answer_str,
        string memory _correct_ans_reason)
```

(d) To assign the winnings for each quiz questions: Based on answers chosen by users while participating, the users with correct answers predicted are assigned with the winnings.

The winnings are calculated as:

- For Integer type: The user with the closest prediction of the match answer gets all the participation fee collected. If there is more than 1 user with the same answer, winnings are distributed equally.
- For MCQ type: The users with the correct MCQ option selected gets the winnings. All the users who selected other options, the participation fee collected from those users in total is distributed equally amongst the winners.

```
function getQidWinning(
        uint256 _qid,
        address participant)
```

(e) Fetch the winnings of any address publicly: Anyone should be able to fetch the total winnings using their account address for transparency.

function getWinnningBalance(address from) public view returns (uint256)

(f) Retrieve the winnings by users: When the balance of platform tokens is transferred to smart contract, then the mobile app can give FIAT winnings to users after cutting the platform fees.

We can create an oracle for sources of truth from sports APIs in the future which can be run by nodes independently.

Combining all the requirements and implementing them as smart contract, we get the following smart contract:

Smart contract codes

SportsFantasy.sol

```
pragma solidity >=0.6.0 <0.8.0;
pragma experimental ABIEncoderV2;

// SPDX-License-Identifier: MIT
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "./Initializable.sol";
import "./EIP712MetaTxnUpg.sol";

contract SportsFantasy is Initializable, EIP712MetaTxnUpg {
  using SafeMath for uint256;

  // This is a type for a single MCQ Option.
  struct Option {
     uint8 name; // short name of mcq text
     uint256 voteCount; // number of accumulated votes
     address[] voters; // all voters who voted for particular option
  }

  // participant->qid->ansid->bool
  mapping(address => mapping(uint256 => mapping(uint256 => bool))) optionsSelected;

  // participant->qid->bool
  mapping(address => mapping(uint256 => bool)) rewardProvided;

  struct Question {
     address author;
     uint256 _type; // 2=INT, 1=MCQ type
     string qcontent; // content
     uint256 vote_price; // price/rate for 1 vote
     uint256 correct_ans_id;
```

```
    Option[6] options; // all max 6 options
    uint256 totalCount;
    address[] winners;
    string[] winnersnames;
    string correct_answer_str;
    string correct_ans_reason;
    uint256 distributed_each;
    uint256 totalWinners;
    bool status; // true for live question
    bool stopped;
    bool declared_result;
}

address public feeAddress; //=0x397B73151D8Ee4D4B66741E49744ed1BDAB95fe9;

IERC20 sdToken;

struct Participant {
    address _add;
    uint256 _points;
}
mapping(uint256 => Participant[]) allContestParticipants;

// sorted by rank
mapping(uint256 => Participant[]) rankParticipants;

mapping(address => uint256) public balanceOf;

// for all questions
mapping(uint256 => Question) questions;

mapping(address => string) naam;

// contribution/ditribution
event FundTransfer(address backer, uint256 amount, bool isContribution);

event QuestionCreation(address backer, uint256 qid, uint256 _type);

event VotingDone(address backer, uint256 qid, uint256 ansid);

event WinnerDeclared(
    uint256 qid,
    string name,
    address winner,
    uint256 rewardamt,
    uint256 ans,
    uint256 _type
);

function initialize(address owner, address erc20Token) public initializer {
    sdToken = IERC20(erc20Token);
    manager = owner;
    feeAddress = 0x397B73151D8Ee4D4B66741E49744ed1BDAB95fe9;
    EIP712MetaTxnUpg.__EIP712MetaTxnUpg_init("SportsDapp", "");
```

```
}

//_type; // 1=mcq, 2=integer type
function postQuestion(
    uint256 _qid,
    string memory _qcontent,
    uint256 _vote_price,
    uint8 _optionslength,
    uint256 _type,
    string memory name
) public {
    require(questions[_qid].status != true, "Question id already exists");

    naam[msgSender()] = name;

    if (_type == 2) {
        for (uint8 i = 0; i < _optionslength; i++) {
            questions[_qid].options[i].name = i;
            questions[_qid].options[i].voteCount = 0;
        }
        questions[_qid]._type = 2;
    } else if (_type == 1) questions[_qid]._type = 1;
    questions[_qid].author = msgSender();
    questions[_qid].qcontent = _qcontent;
    questions[_qid].vote_price = _vote_price;
    questions[_qid].totalCount = 0;
    questions[_qid].status = true;
    questions[_qid].declared_result = false;
    questions[_qid].stopped = false;
    emit QuestionCreation(msgSender(), _qid, _type);
}

function stopVoting(uint256 _qid) public onlyAdmin {
    require(
        questions[_qid].stopped != true,
        "Voting process has been already stopped"
    );
    questions[_qid].stopped = true;
}

function getQidWinning(uint256 _qid, address participant)
    public
    returns (uint256)
{
    require(
        questions[_qid].declared_result == true,
        "Result not yet declared"
    );
    require(
        rewardProvided[participant][_qid] == false,
        "Reward already provided"
    );

    uint256 ans_id = questions[_qid].correct_ans_id;
```

```
uint256 totalreward = questions[_qid].distributed_each;

if (ans_id == 999) {
   for (uint256 optionno = 0; optionno < 6; optionno++) {
      if (optionsSelected[participant][_qid][optionno] == true) {
         sdToken.transfer(participant, totalreward);
         questions[_qid].winners.push(participant);
         questions[_qid].winnersnames.push(naam[participant]);
         emit WinnerDeclared(
            _qid,
            naam[participant],
            participant,
            totalreward,
            ans_id,
            1
         );
         rewardProvided[participant][_qid] = true;
         balanceOf[participant] += totalreward;
         return totalreward;
      }
   }
} else {
   uint256 winningCount = questions[_qid].options[ans_id].voteCount;
   if (winningCount == 0) {
      for (uint256 optionno = 0; optionno < 6; optionno++) {
         if (optionsSelected[participant][_qid][optionno] == true) {
            sdToken.transfer(participant, totalreward);
            questions[_qid].winners.push(participant);
            questions[_qid].winnersnames.push(naam[participant]);
            emit WinnerDeclared(
               _qid,
               naam[participant],
               participant,
               totalreward,
               ans_id,
               1
            );
            rewardProvided[participant][_qid] = true;
            balanceOf[participant] += totalreward;
            return totalreward;
         }
      }
   } else if (optionsSelected[participant][_qid][ans_id] == true) {
      sdToken.transfer(participant, totalreward);
      questions[_qid].winners.push(participant);
      questions[_qid].winnersnames.push(naam[participant]);

      emit WinnerDeclared(
         _qid,
         naam[participant],
         participant,
         totalreward,
         ans_id,
         1
```

```
            );
            rewardProvided[participant][_qid] = true;
            balanceOf[participant] += totalreward;

            return totalreward;
        }
    }

    return 0;
}

// ans id in case of mcq and integer in case of integer type question
function submitRightSolution(
    uint256 _qid,
    uint256 _ansid,
    string memory _correct_answer_str,
    string memory _correct_ans_reason
) public onlyAdmin {
    // based on live feed this should be updated

    // require(questions[_qid].declared_result!=1,"Right solution declared already");

    questions[_qid].correct_ans_id = _ansid;
    questions[_qid].declared_result = true;
    questions[_qid].stopped = true;
    questions[_qid].correct_answer_str = _correct_answer_str;
    questions[_qid].correct_ans_reason = _correct_ans_reason;

    uint256 totalreward;

    // none answered correct or INVALID, send back fee to all
    if (_ansid == 999) {
        totalreward = (questions[_qid].vote_price);
        questions[_qid].distributed_each = totalreward;
    } else {
        uint256 winningCount = questions[_qid].options[_ansid].voteCount;
        if (winningCount == 0) {
            totalreward = (questions[_qid].vote_price);
            questions[_qid].distributed_each = totalreward;
        }
        // no fees if participants<=2 else 2.5% fee
        else if (questions[_qid].options[_ansid].voters.length <= 2) {
            uint256 totalCount = questions[_qid].totalCount;
            questions[_qid].totalWinners = winningCount;
            uint256 part = totalCount.div(winningCount);
            totalreward = part.mul(questions[_qid].vote_price);

            questions[_qid].distributed_each = totalreward;
        } else {
            uint256 totalCount = questions[_qid].totalCount;
            questions[_qid].totalWinners = winningCount;
            uint256 part = totalCount.div(winningCount);
            totalreward = part.mul(questions[_qid].vote_price);
```

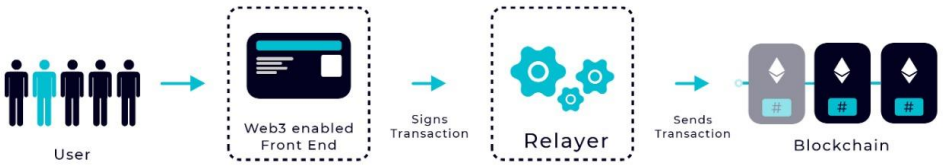
```
            uint256 totalfee = totalreward / 40;
            uint256 totalwin = totalreward - totalfee;
            questions[_qid].distributed_each = totalwin;
        }
    }
}

// ans id in case of mcq and integer in case of integer type question
function participate(
    uint256 _qid,
    uint256 ansid,
    string memory name
) public {
    // allowed 1 time only for particular acct
    // not allowed after solution given

    require(
        questions[_qid].declared_result != true,
        "Voting process has been declared"
    );
    require(
        questions[_qid].stopped != true,
        "Voting process has been stopped"
    );
    uint256 _amount = questions[_qid].vote_price;

    sdToken.transferFrom(msgSender(), address(this), _amount);

    naam[msgSender()] = name;

    questions[_qid].options[ansid].voters.push(msgSender());
    questions[_qid].options[ansid].voteCount += 1;
    optionsSelected[msgSender()][_qid][ansid] = true;

    questions[_qid].totalCount += 1;
    emit VotingDone(msgSender(), _qid, ansid);
}

// total winning of the account
function getWinnningBalance(address from) public view returns (uint256) {
    return balanceOf[from];
}

// get Matic balance
function getBalance(address from) public view returns (uint256) {
    return (address(from).balance);
}

// get winners list
function getWinners(uint256 _qid) public view returns (address[] memory) {
    return questions[_qid].winners;
}

function getWinnersAnsStr(uint256 _qid)
```

```
    public
    view
    returns (string memory)
{
    return (questions[_qid].correct_answer_str);
}

function getWinnersAnsReason(uint256 _qid)
    public
    view
    returns (string memory)
{
    return (questions[_qid].correct_ans_reason);
}

function getWinnersDist(uint256 _qid) public view returns (uint256) {
    return (questions[_qid].distributed_each);
}

function getWinnersno(uint256 _qid) public view returns (uint256) {
    return (questions[_qid].totalWinners);
}

function getRate(uint256 _qid) public view returns (uint256) {
    return questions[_qid].vote_price;
}

function getQcontent(uint256 _qid) public view returns (string memory) {
    return questions[_qid].qcontent;
}

function getQ(uint256 _qid) public view returns (Question memory) {
    return questions[_qid];
}

address public manager; // address used to set Admins
address[] public Admins;
mapping(address => bool) public AdminByAddress;

event SetAdmins(address[] Admins);

modifier onlyAdmin() {
    require(AdminByAddress[msgSender()] == true);
    _;
}

/**
 * @dev MultiOwnable constructor sets the manager
 */
// constructor(address _Admin) {
//     require(_Admin != address(0), "Admin address cannot be 0");
//     manager = _Admin;
// }
```

```
  /**
   * @dev Function to set Admins addresses
   */
  function setAdmins(address[] memory _Admins) public {
      require(msgSender() == manager);
      _setAdmins(_Admins);
  }

  function _setAdmins(address[] memory _Admins) internal {
      for (uint256 i = 0; i < Admins.length; i++) {
          AdminByAddress[Admins[i]] = false;
      }

      for (uint256 j = 0; j < _Admins.length; j++) {
          AdminByAddress[_Admins[j]] = true;
      }
      Admins = _Admins;
      emit SetAdmins(_Admins);
  }

  function getAdmins() public view returns (address[] memory) {
      return Admins;
  }

  modifier onlyManager() {
      _onlyManager();
      _;
  }

  function _onlyManager() private view {
      require(
          msgSender() == manager,
          "Only the contract manager may perform this action"
      );
  }
}
```

In the above SportsFantasy contract, we see the import line
import "./EIP712MetaTxnUpg.sol";

We are using meta transaction in the DApp. With this it doesn't matter which relayer relays your transaction as our code will be independent of msg.sender property.



Meta Transaction Flow

This standard from Biconomy allows user's wallets to display data in signing prompts in a structured and readable format. EIP712 is a great step forward for security and usability. Users participating in app will no longer need to sign off on inscrutable hexadecimal strings, which is confusing & insecure.

Following is the full code of EIP712MetaTxn.sol dependencies:

```
// SPDX-License-Identifier: MIT
pragma solidity 0.7.0;
import "./EIP712BaseUpg.sol";
import "@openzeppelin/contracts/math/SafeMath.sol";

contract EIP712MetaTxnUpg is EIP712BaseUpg {
  using SafeMath for uint256;
  bytes32 private constant META_TRANSACTION_TYPEHASH = keccak256(bytes("MetaTransaction(uint256
nonce,address from,bytes functionSignature)"));

  event MetaTransactionExecuted(address userAddress, address payable relayerAddress, bytes functionSignature);
  mapping(address => uint256) private nonces;

  /*
   * Meta transaction structure.
   * No point of including value field here as if user is doing value transfer then he has the funds to pay for gas
   * He should call the desired function directly in that case.
   */
  struct MetaTransaction {
    uint256 nonce;
    address from;
    bytes functionSignature;
  }

  /**
   * @dev Initializes the contract setting the deployer as the initial owner.
   */
  function __EIP712MetaTxnUpg_init(string memory name, string memory version) internal initializer {
    __EIP712BaseUpg_init_unchained(name,version);
  }
```

```
  function __EIP712MetaTxnUpg_init_unchained(string memory name, string memory version) internal initializer {
    __EIP712BaseUpg_init_unchained(name,version);
  }

  function convertBytesToBytes4(bytes memory inBytes) internal pure returns (bytes4 outBytes4) {
    if (inBytes.length == 0) {
      return 0x0;
    }

    assembly {
      outBytes4 := mload(add(inBytes, 32))
    }
  }

  function executeMetaTransaction(address userAddress,
    bytes memory functionSignature, bytes32 sigR, bytes32 sigS, uint8 sigV) public payable returns(bytes memory) {
    bytes4 destinationFunctionSig = convertBytesToBytes4(functionSignature);
    require(destinationFunctionSig != msg.sig, "functionSignature can not be of executeMetaTransaction method");
    MetaTransaction memory metaTx = MetaTransaction({
      nonce: nonces[userAddress],
      from: userAddress,
      functionSignature: functionSignature
    });
    require(verify(userAddress, metaTx, sigR, sigS, sigV), "Signer and signature do not match");
    nonces[userAddress] = nonces[userAddress].add(1);
    // Append userAddress at the end to extract it from calling context
    (bool success, bytes memory returnData) = address(this).call(abi.encodePacked(functionSignature, userAddress));

    require(success, "Function call not successful");
    emit MetaTransactionExecuted(userAddress, msg.sender, functionSignature);
    return returnData;
  }

  function hashMetaTransaction(MetaTransaction memory metaTx) internal pure returns (bytes32) {
    return keccak256(abi.encode(
      META_TRANSACTION_TYPEHASH,
      metaTx.nonce,
      metaTx.from,
      keccak256(metaTx.functionSignature)
    ));
  }

  function getNonce(address user) external view returns(uint256 nonce) {
    nonce = nonces[user];
  }

  function verify(address user, MetaTransaction memory metaTx, bytes32 sigR, bytes32 sigS, uint8 sigV) internal view
returns (bool) {
    address signer = ecrecover(toTypedMessageHash(hashMetaTransaction(metaTx)), sigV, sigR, sigS);
    require(signer != address(0), "Invalid signature");
    return signer == user;
  }
```

```
  function msgSender() internal view returns(address sender) {
    if(msg.sender == address(this)) {
      bytes memory array = msg.data;
      uint256 index = msg.data.length;
      assembly {
        // Load the 32 bytes word from memory with the address on the lower 20 bytes, and mask those.
        sender := and(mload(add(array, index)), 0xffffffffffffffffffffffffffffffffffffffff)
      }
    } else {
      sender = msg.sender;
    }
    return sender;
  }
}


pragma solidity >=0.6.0 <0.8.0;
//SPDX-License-Identifier: MIT
import "./Initializable.sol";

contract EIP712BaseUpg is Initializable{

  struct EIP712Domain {
    string name;
    string version;
    uint256 salt;
    address verifyingContract;
  }

  bytes32 internal constant EIP712_DOMAIN_TYPEHASH = keccak256(bytes("EIP712Domain(string name,string
version,uint256 salt,address verifyingContract)"));

  bytes32 internal domainSeperator;

  /**
   * @dev Initializes the contract setting the deployer as the initial owner.
   */
  function __EIP712BaseUpg_init(string memory name, string memory version) internal initializer {
  }

  function __EIP712BaseUpg_init_unchained(string memory name, string memory version) internal initializer {
    domainSeperator = keccak256(abi.encode(
      EIP712_DOMAIN_TYPEHASH,
      keccak256(bytes(name)),
      keccak256(bytes(version)),
      getChainID(),
      address(this)
    ));

  }
  function getChainID() internal pure returns (uint256 id) {
    assembly {
      id := chainid()
    }
```

```
  }

  function getDomainSeperator() private view returns(bytes32) {
    return domainSeperator;
  }

  /**
  * Accept message hash and returns hash message in EIP712 compatible form
  * So that it can be used to recover signer from signature signed using EIP712 formatted data
  * https://eips.ethereum.org/EIPS/eip-712
  * "\\x19" makes the encoding deterministic
  * "\\x01" is the version byte to make it compatible to EIP-191
  */
  function toTypedMessageHash(bytes32 messageHash) internal view returns(bytes32) {
    return keccak256(abi.encodePacked("\x19\x01", getDomainSeperator(), messageHash));
  }

}
```

3. PYTHON SCRIPTS TO FETCH MATCH DETAILS

We need multiple python scripts to fetch upcoming matches, create auto generated questions, fetch match results based on match outcomes, send the transactions on blockchain & using the APIs.

Types of Questions

Types of Cricket MCQ questions:

- To Win the Match
- How many runs will be scored in 1st inning
- To Win the Toss
- Player of the match
- 1st Over Total Runs
- 1st Wicket Method? Options as Caught / Bowled / LBW / Run Out / Stumped / Others
- Most Match Sixes Team
- Most Match Fours Team
- Most Run Outs (Fielding) Team
- Runs at Fall of 1st Wicket
- A Hundred to be Scored in the Match? Yes/No
- Opening Partnership Total
- Highest Partnership by which team
- Who will score maximum runs in the match from Team A
- Who will take maximum wickets economically from Team A
- Who will have the best economy from Team A
- Who will score maximum runs in the match from Team B
- Who will take maximum wickets economically from Team B
- Who will have the best economy from Team B
- How many runs will Player A score? Range of runs as options
- How many wickets will Bowler B take?

Types of Football MCQ questions:

- What will be the result of the match
- What will be total goals made in the match
- Which team will lead in half-time (first session)
- Which team will lead in half-full time (second session)
- In which category total goals scored in the match falls? Odd/Even
- Which team will score first goal in the match
- What will be total goals made till half-time (first session)
- What will be total goals made in half-full time (second session)
- How many goals Team A will score?
- How many goals Team B will score?

Respective options are given for each question for user to choose from. Each question may have different times when result is declared. eg. The toss result can be declared before match starts, 1st over runs can be fetched after over finishes, Runs ar Fall of 1st wicket, method of 1st wicket can be fetched after 1st wicket falls.

Types of Cricket Integer questions:

- How many runs will be scored in 1st inning
- 1st Over Total Runs
- Runs at Fall of 1st Wicket
- Opening Partnership Total
- How many runs will Player A score?

Types of Football Integer questions:

- What will be total goals made in the match
- What will be total goals made till half-time (first session)
- What will be total goals made in half-full time (second session)
- How many goals Team A will score?
- How many goals Team B will score?

Based on integers input by user, the user/users with closest input entered gets the winnings distributed equally.

Each question item in an array created in python script has following identifiers

```
"que": "To Win the Toss",
"qno": "2",
"teamid": "",
"playerid": "",
"type": "1",
"oplen": 2,
"before": "Before a hour the match starts",
"options": [“Team A”, “Team B”]
```

Each of these identifier describes the question type. It includes question content, question number, team id related in question, player id related in question, MCQ/Integer type, options length for MCQ types, validity of question before when user is allowed to answer & options. The participation fee for these auto generated questions is kept 1 or 2 tokens based on importance of question. Any user can create private room and select the questions from above list. The private room ID can be shared within private groups to play in closed group at any participation fees.

The scripts to create questions for cricket & football keeps running independently. We can use screen function in server terminal to keep the python codes running. As soon as new match id is returned from the Sportsmonk upcoming matches API, new questions are created for that match id and stored in MongoDB from where mobile app fetches the questions.

Another scripts keep running to try and find the answers of questions generated automatically or for private rooms. Based on identifiers/metadata on question, Sportsmonk APIs can be used to fetch the required results.

Python Functions

Python functions are created to fetch raw data from Sportsmonk APIs. The functions are added below:

```
  def livematches(self):
     print('api call,livematches')
     today = date.today()
     url = "https://cricket.sportmonks.com/api/v2.0/fixtures?" \
          "api_token=h5Ab6JWRICYXpmB7TiRxlBOvvEDLWe3uiSQxvSiRI1mCMUJG1DhfUsVozqZn" \
          "&filter[starts_between]="+str(today)+","+str(today+timedelta(days=2))+\
      "&include=localteam,visitorteam,batting,bowling,lineup,scoreboards.team,league&sort=starting_at"
     res = requests.request("GET", url)
     print(res.text)
     if res.status_code == 200 and res:
        return json.loads(res.text)


  def getSquad(self, id, sid):
     print('api call, getsquad1')
     url = "https://cricket.sportmonks.com/api/v2.0/teams/"+str(id)+"/squad/"+str(sid)+"?" \
        "api_token=h5Ab6JWRICYXpmB7TiRxlBOvvEDLWe3uiSQxvSiRI1mCMUJG1DhfUsVozqZn"
     res = requests.request("GET", url)
     if res.status_code == 200 and res:
        return json.loads(res.text)


  def playerStatistics(self,pid):
     print('API Call')
     url =
"https://cricket.sportmonks.com/api/v1/players/"+str(pid)+"?api_token=h5Ab6JWRICYXpmB7TiRxlBOvvEDLWe3uiSQx
vSiRI1mCMUJG1DhfUsVozqZn&include=career&season=10"
     res = requests.request("GET",url)
     if res.status_code == 200 and res:
        #print(res.text)
        return json.loads(res.text)


  def Players(self, mid, sid):
     print('API call')
     url = "https://cricket.sportmonks.com/api/v2.0/fixtures/"+str(mid)+"?" \
"api_token=h5Ab6JWRICYXpmB7TiRxlBOvvEDLWe3uiSQxvSiRI1mCMUJG1DhfUsVozqZn&include=lineup,localteam,visi
torteam"
     res = requests.request("GET", url)
     if res.status_code == 200 and res:
        print(res.text)
        return json.loads(res.text)
```

```
  def scorecard(self,mid):
     print('api call, match id')
     url = "https://cricket.sportmonks.com/api/v2.0/fixtures/" + str(mid) + "?" \
"api_token=h5Ab6JWRICYXpmB7TiRxlBOvvEDLWe3uiSQxvSiRI1mCMUJG1DhfUsVozqZn&include=lineup,localteam,visi
torteam"
     try:
        res = requests.request("GET", url)
        if res.status_code == 200 and res:
           return json.loads(res.text)
     except:
        print('scoreboard:connection refused')
```

The python code is written to transform data fetched in these functions and stored in off chain DBs in particular format.

Creating MCQ or INT questions in python script using Sportsmonk API:

```
  def Q_generation(self, mid, data, client):
     q_arr = [{"que": "To Win the Match", "qno": "1", "teamid": "", "playerid": "", "type": "1", "oplen": 2,
              "before": "Before the match starts",
              "options": json.dumps([data["localteam"]["name"], data["visitorteam"]["name"], "Draw"])},
             {"que": "To Win the Toss", "qno": "2", "teamid": "", "playerid": "", "type": "1", "oplen": 2,
              "before": "Before a hour the match starts",
              "options": json.dumps([data["localteam"]["name"], data["visitorteam"]["name"]])}]

     batter1 = []
     batter2 = []
     bowler1 = []
     bowler2 = []
     all_rounder1 = []
     all_rounder2 = []
     team1_id = data['localteam_id']
     team2_id = data['visitorteam_id']
     team1 = self.getSquad(data['localteam_id'], data['season_id'])
     if team1['data']['squad']==[]:
        team1 = self.getSquad2(data['localteam_id'])
     team2 = self.getSquad(data['visitorteam_id'], data['season_id'])
     if team2['data']['squad']==[]:
        team2 = self.getSquad2(data['visitorteam_id'])



     for player in team1['data']['squad']:
        seasons = self.playerStatistics(player['id'])

        credit = None
        if 'included' in seasons:
           found = False
           for s in seasons['included']:
              if s['attributes']['season_id']==data['season_id']:
```

```
                if player['position']['name'] in ["Batsman","Wicketkeeper","Allrounder"]:
                    print(player['id'])
                    if s['attributes']['batting']:
                        avg =s['attributes']['batting']['average']
                        if avg>=40 or (s['attributes']['batting']['strike_rate'] and s['attributes']['batting']['strike_rate']>=120
and avg>=30):
                            credit = 9.0
                        elif 30<=avg<40:
                            credit = 8.5
                        elif 25<=avg<30:
                            credit = 8.0
                        elif 20<=avg<25:
                            credit = 7.5
                        elif 15<=avg<20:
                            credit = 7.0
                        elif 10<=avg<15:
                            credit = 6.5
                        else:
                            credit =6.0

                    if player['position']['name'] in ["Bowler","Allrounder"]:
                        if s['attributes']['bowling']:
                            avg = s['attributes']['bowling']['average']

                            if avg<=15:
                                credit1 = 9.0
                            elif 15<avg<=20:
                                credit1 = 8.5
                            elif 20<avg<=25:
                                credit1 = 8.0
                            elif 25<=avg<30:
                                credit1 = 7.5
                            elif 30<=avg<40:
                                credit1 = 7.0
                            elif 40<=avg<45:
                                credit1 = 6.5
                            else:
                                credit1 = 6.0

                            if credit:
                                if credit1>=8:
                                    credit = credit1
                                elif credit<8:
                                    credit = (credit+credit1)/2
                            else:
                                credit = credit1
                        found = True
                        break

            if found == False:
                for s in seasons['included']:
                    if s['attributes']['tournament_type']==data['type']:
```

```
                if player['position']['name'] in ["Batsman","Wicketkeeper","Allrounder"]:
                    if s['attributes']['batting']:
                        avg =s['attributes']['batting']['average']
                        if avg>=40 or (s['attributes']['batting']['strike_rate'] and s['attributes']['batting']['strike_rate']>=120
and avg>=30):
                            credit = 9.0
                        elif 30<=avg<40:
                            credit = 8.5
                        elif 25<=avg<30:
                            credit = 8.0
                        elif 20<=avg<25:
                            credit = 7.5
                        elif 15<=avg<20:
                            credit = 7.0
                        elif 10<=avg<15:
                            credit = 6.5
                        else:
                            credit =6.0

                if player['position']['name'] in ["Bowler","Allrounder"]:
                    if s['attributes']['bowling']:
                        avg = s['attributes']['bowling']['average']

                        if avg<=15:
                            credit1 = 9.0
                        elif 15<avg<=20:
                            credit1 = 8.5
                        elif 20<avg<=25:
                            credit1 = 8.0
                        elif 25<=avg<30:
                            credit1 = 7.5
                        elif 30<=avg<40:
                            credit1 = 7.0
                        elif 40<=avg<45:
                            credit1 = 6.5
                        else:
                            credit1 = 6.0

                        if credit:
                            if credit1>=8:
                                credit = credit1
                            elif credit<8:
                                credit = (credit+credit1)/2
                        else:
                            credit = credit1
                found = True
                break

        player['credit']=credit if credit!=None else 6.0

        if player['position']['name'] == 'Batsman':
            batter1.append(player['fullname'])

        elif player['position']['name'] == 'Bowler':
```

```
            bowler1.append(player['fullname'])

        else:
            all_rounder1.append(player['fullname'])

    for player in team2['data']['squad']:
        seasons = self.playerStatistics(player['id'])

        credit = None
        if 'included' in seasons:
            found = False
            for s in seasons['included']:
                if s['attributes']['season_id']==data['season_id']:

                    if player['position']['name'] in ["Batsman","Wicketkeeper","Allrounder"]:
                        if s['attributes']['batting']:
                            avg =s['attributes']['batting']['average']
                            if avg>=40 or (s['attributes']['batting']['strike_rate'] and s['attributes']['batting']['strike_rate']>=120
and avg>=30):
                                credit = 9.0
                            elif 30<=avg<40:
                                credit = 8.5
                            elif 25<=avg<30:
                                credit = 8.0
                            elif 20<=avg<25:
                                credit = 7.5
                            elif 15<=avg<20:
                                credit = 7.0
                            elif 10<=avg<15:
                                credit = 6.5
                            else:
                                credit =6.0

                    if player['position']['name'] in ["Bowler","Allrounder"]:
                        if s['attributes']['bowling']:
                            avg = s['attributes']['bowling']['average']

                            if avg<=15:
                                credit1 = 9.0
                            elif 15<avg<=20:
                                credit1 = 8.5
                            elif 20<avg<=25:
                                credit1 = 8.0
                            elif 25<=avg<30:
                                credit1 = 7.5
                            elif 30<=avg<40:
                                credit1 = 7.0
                            elif 40<=avg<45:
                                credit1 = 6.5
                            else:
                                credit1 = 6.0

                            if credit:
                                if credit1>=8:
```

```
                        credit = credit1
                    elif credit<8:
                        credit = (credit+credit1)/2
                else:
                    credit = credit1
            found = True
            break

    if found == False:
        for s in seasons['included']:
            if s['attributes']['tournament_type']==data['type']:

                if player['position']['name'] in ["Batsman","Wicketkeeper","Allrounder"]:
                    if s['attributes']['batting']:
                        avg =s['attributes']['batting']['average']
                        if avg>=40 or (s['attributes']['batting']['strike_rate'] and s['attributes']['batting']['strike_rate']>=120
and avg>=30):
                            credit = 9.0
                        elif 30<=avg<40:
                            credit = 8.5
                        elif 25<=avg<30:
                            credit = 8.0
                        elif 20<=avg<25:
                            credit = 7.5
                        elif 15<=avg<20:
                            credit = 7.0
                        elif 10<=avg<15:
                            credit = 6.5
                        else:
                            credit =6.0

                if player['position']['name'] in ["Bowler","Allrounder"]:
                    if s['attributes']['bowling']:
                        avg = s['attributes']['bowling']['average']

                        if avg<=15:
                            credit1 = 9.0
                        elif 15<avg<=20:
                            credit1 = 8.5
                        elif 20<avg<=25:
                            credit1 = 8.0
                        elif 25<=avg<30:
                            credit1 = 7.5
                        elif 30<=avg<40:
                            credit1 = 7.0
                        elif 40<=avg<45:
                            credit1 = 6.5
                        else:
                            credit1 = 6.0

                        if credit:
                            if credit1>=8:
                                credit = credit1
                            elif credit<8:
```

```
                        credit = (credit+credit1)/2
                    else:
                        credit = credit1
                found = True
                break

        player['credit']=credit if credit!=None else 6.0

        if player['position']['name'] == 'Batsman':
            batter2.append(player['fullname'])

        elif player['position']['name'] == 'Bowler':
            bowler2.append(player['fullname'])

        else:
            all_rounder2.append(player['fullname'])


client.CRICKETUPCOMINGLIVE['scorecard'].update({'_id':str(mid)},{'$set':{'localteamsquad':team1,'visitorteamsquad':team2}})

    q_arr.append({"que": "Player of the match","qno": "9",
            "oplen": len(batter1)+len(bowler1)+len(all_rounder1)+len(batter2)+len(bowler2)+len(all_rounder2), "teamid": "", "playerid": "", "type": "1", "before": "Before the match starts","options": json.dumps(batter1+bowler1+all_rounder1+batter2+bowler2+all_rounder2)})

    q_arr.append({"que": "1st Over Total Runs","qno": "10",
            "oplen": 2, "teamid": "", "playerid": "", "type": "1", "before": "Before the match starts","options": json.dumps(["Over 1.5","Under 1.5"])})
    q_arr.append({"que": "1st Wicket Method","qno": "11",
            "oplen": 6, "teamid": "", "playerid": "", "type": "1", "before": "Before the match starts","options": json.dumps(["Caught","Bowled","LBW","Run Out","Stumped","Others"])})
    q_arr.append({"que": "Most Match Sixes","qno": "12",
            "oplen": 3, "teamid": "", "playerid": "", "type": "1", "before": "Before the match starts","options": json.dumps([data["localteam"]["name"],"Tie",data["visitorteam"]["name"]])})
    q_arr.append({"que": "Most Match Fours","qno": "13",
            "oplen": 3, "teamid": "", "playerid": "", "type": "1", "before": "Before the match starts","options": json.dumps([data["localteam"]["name"],"Tie",data["visitorteam"]["name"]])})
    q_arr.append({"que": "Most Run Outs (Fielding)","qno": "14",
            "oplen": 3, "teamid": "", "playerid": "", "type": "1", "before": "Before the match starts","options": json.dumps([data["localteam"]["name"],"Tie",data["visitorteam"]["name"]])})
    q_arr.append({"que": "Runs at Fall of 1st Wicket","qno": "15",
            "oplen": 2, "teamid": "", "playerid": "", "type": "1", "before": "Before the match starts","options": json.dumps(["Under 28.5","Over 28.5"])})
    q_arr.append({"que": "A Hundred to be Scored in the Match","qno": "16",
            "oplen": 2, "teamid": "", "playerid": "", "type": "1", "before": "Before the match starts","options": json.dumps(["No","Yes"])})
    q_arr.append({"que":data["localteam"]["name"]+ " Opening Partnership Total","qno": "17", "oplen": 2, "teamid": str(team1_id), "playerid": "", "type": "1", "before": "Before the match starts","options": json.dumps(["Over 26.5","Under 26.5"])})
    q_arr.append({"que":data["visitorteam"]["name"]+ " Opening Partnership Total","qno": "17", "oplen": 2, "teamid": str(team2_id), "playerid": "", "type": "1", "before": "Before the match starts","options": json.dumps(["Over 26.5","Under 26.5"])})
```

```
    if len(batter1) >= 6:
        q_arr.append(
            {"que": "Who will score maximum runs in the match from {}?".format(data["localteam"]["name"]),"qno": "3",
             "oplen": 6, "teamid": str(team1_id), "playerid": "", "type": "1", "before": "Before the match starts","options":
json.dumps(batter1[:6])})
        q_arr.append(
            {"que": "Who's strike rate will be best from {}?".format(data["localteam"]["name"]), "qno": "4","oplen": 6,
"teamid": str(team1_id),
             "playerid": "", "type": "1", "before": "Before the match starts", "options": json.dumps(batter1[:6])})
    else:
        q_arr.append(
            {"que": "Who will score maximum runs in the match from {}?".format(data["localteam"]["name"]),
             "qno": "3","oplen": 6, "teamid": str(team1_id), "playerid": "", "type": "1", "before": "Before the match starts",
             "options": json.dumps(batter1+all_rounder1[:6-len(batter1)])})
        q_arr.append(
            {"que": "Who's strike rate will be best from {}?".format(data["localteam"]["name"]), "qno": "4",
             "oplen": 6, "teamid": str(team1_id),"playerid": "", "type": "1", "before": "Before the match starts", "options":
json.dumps(batter1+all_rounder1[:6-len(batter1)])})

    if len(batter2) >= 6:
        q_arr.append(
            {"que": "Who will score maximum runs in the match from {}?".format(data["visitorteam"]["name"]),
             "qno": "3","oplen": 6, "teamid": str(team2_id), "playerid": "", "type": "1", "before": "Before the match starts",
             "options": json.dumps(batter2[:6])})
        q_arr.append(
            {"que": "Who's strike rate will be best from {}?".format(data["visitorteam"]["name"]), "qno": "4",
             "oplen": 6, "teamid": str(team2_id), "playerid": "", "type": "1", "before": "Before the match starts","options":
json.dumps(batter2[:6])})
    else:
        q_arr.append(
            {"que": "Who will score maximum runs in the match from {}?".format(data["visitorteam"]["name"]),
             "qno": "3", "oplen": 6, "teamid": str(team2_id), "playerid": "", "type": "1",
             "before": "Before the match starts","options": json.dumps(batter2+all_rounder2[:6-len(batter2)])})
        q_arr.append(
            {"que": "Who's strike rate will be best from {}?".format(data["visitorteam"]["name"]), "qno": "4",
             "oplen": 6, "teamid": str(team2_id), "playerid": "", "type": "1", "before": "Before the match starts",
             "options": json.dumps(batter2+all_rounder2[:6-len(batter2)])})

    if len(bowler1) >= 6:
        q_arr.append(
            {"que": "Who will take maximum wickets economically from {}".format(data["localteam"]["name"]),
             "qno": "5", "oplen": 6, "teamid": str(team1_id), "playerid": "", "type": "1", "before": "Before the match
starts",
             "options": json.dumps(bowler1[:6])})
        q_arr.append(
            {"que": "Who will have the best economy from {}".format(data["localteam"]["name"]), "qno": "6",
             "oplen": 6, "teamid": str(team1_id), "playerid": "", "type": "1", "before": "Before the match starts",
             "options": json.dumps(bowler1[:6])})
    else:
        q_arr.append(
            {"que": "Who will take maximum wickets economically from {}".format(data["localteam"]["name"]),
             "qno": "5","oplen": 6, "teamid": str(team1_id), "playerid": "", "type": "1", "before": "Before the match starts",
             "options": json.dumps(bowler1+all_rounder1[:6-len(bowler1)])})
        q_arr.append(
```

```
            {"que": "Who will have the best economy from {}".format(data["localteam"]["name"]), "qno": "6",
             "oplen": 6, "teamid": str(team1_id), "playerid": "", "type": "1", "before": "Before the match starts",
             "options": json.dumps(bowler1+all_rounder1[:6-len(bowler1)])})

    if len(bowler2) >= 6:
        q_arr.append(
            {"que": "Who will take maximum wickets economically from {}".format(data["visitorteam"]["name"]),
             "qno": "5", "oplen": 6, "teamid": str(team2_id), "playerid": "", "type": "1", "before": "Before the match
starts",
             "options": json.dumps(bowler2[:6])})
        q_arr.append(
            {"que": "Who will have the best economy from {}".format(data["visitorteam"]["name"]), "qno": "6",
             "oplen": 6, "teamid": str(team2_id), "playerid": "", "type": "1", "before": "Before the match starts",
             "options": json.dumps(bowler2[:6])})
    else:
        q_arr.append(
            {"que": "Who will take maximum wickets economically from {}".format(data["visitorteam"]["name"]),
             "qno": "5", "oplen": 6, "teamid": str(team2_id), "playerid": "", "type": "1", "before": "Before the match
starts",
             "options": json.dumps(bowler2+all_rounder2[:6-len(bowler2)])})
        q_arr.append(
            {"que": "Who will have the best economy from {}".format(data["visitorteam"]["name"]), "qno": "6",
             "oplen": 6, "teamid": str(team2_id), "playerid": "", "type": "1", "before": "Before the match starts",
             "options": json.dumps(bowler2+all_rounder2[:6-len(bowler2)])})

    wicketsarr = ['0', '1', '2', '3+']
    runsarr = ['0-20', '21-40', '41-60', '60+']

    for i in team1['data']['squad']:
        if i['position']['name'] in set(['Batsman','Wicketkeeper']):
            q_arr.append({"que": "How many runs will {} score?".format(i["fullname"]), "qno": "7", "oplen": 4, "teamid":
"","playerid": str(i["id"]), "type": "1", "before": "Before the match starts","options": json.dumps(runsarr)})

        elif i['position']['name'] == 'Bowler':
            q_arr.append({"que": "How many wickets will {}
take?".format(i["fullname"]),"qno":"8","oplen":4,"teamid":"","playerid":str(i["id"]),"type":"1","before":"Before the
match starts","options":json.dumps(wicketsarr)})

    for i in team2['data']['squad']:
        if i['position']['name'] in set(['Batsman','Wicketkeeper']):
            q_arr.append({"que": "How many runs will {} score?".format(i["fullname"]), "qno": "7", "oplen": 4, "teamid":
"","playerid": str(i["id"]), "type": "1", "before": "Before the match starts","options": json.dumps(runsarr)})

        elif i['position']['name'] == 'Bowler':
            q_arr.append({"que": "How many wickets will {}
take?".format(i["fullname"]),"qno":"8","oplen":4,"teamid":"","playerid":str(i["id"]),"type":"1","before":"Before the
match starts","options":json.dumps(wicketsarr)})


    return q_arr


  def livematches(self):
    print('api call,livematches')
```

```
        today = date.today()
        url = "https://cricket.sportmonks.com/api/v2.0/fixtures?" \
              "api_token=h5Ab6JWRICYXpmB7TiRxlBOvvEDLWe3uiSQxvSiRI1mCMUJG1DhfUsVozqZn" \
              "&filter[starts_between]="+str(today)+","+str(today+timedelta(days=2))+\
              "&include=localteam,visitorteam,batting,bowling,lineup,scoreboards.team,league&sort=starting_at"

        res = requests.request("GET", url)
        print(res.text)
        if res.status_code == 200 and res:
            return json.loads(res.text)

    def getSquad(self, id, sid):
        print('api call, getsquad1')
        url = "https://cricket.sportmonks.com/api/v2.0/teams/"+str(id)+"/squad/"+str(sid)+"?" \
            "api_token=h5Ab6JWRICYXpmB7TiRxlBOvvEDLWe3uiSQxvSiRI1mCMUJG1DhfUsVozqZn"

        res = requests.request("GET", url)
        if res.status_code == 200 and res:
            return json.loads(res.text)

    def getSquad2(self, id):
        print('API Call,getsquad2')
        url="https://cricket.sportmonks.com/api/v2.0/teams/"+str(id)+"?" \
            "api_token=h5Ab6JWRICYXpmB7TiRxlBOvvEDLWe3uiSQxvSiRI1mCMUJG1DhfUsVozqZn&include=squad"
        try:
            res = requests.request("GET", url)
            if res.status_code == 200 and res:
                return json.loads(res.text)
        except:
            print('getSqaud2:connection refused')

    def playerStatistics(self,pid):
        print('API Call')
        url =
"https://cricket.sportmonks.com/api/v1/players/"+str(pid)+"?api_token=h5Ab6JWRICYXpmB7TiRxlBOvvEDLWe3uiSQx
vSiRI1mCMUJG1DhfUsVozqZn&include=career&season=10"

        res = requests.request("GET",url)
        if res.status_code == 200 and res:
            #print(res.text)
            return json.loads(res.text)

    def Players(self, mid, sid):
        print('API call')
        url = "https://cricket.sportmonks.com/api/v2.0/fixtures/"+str(mid)+"?" \

"api_token=h5Ab6JWRICYXpmB7TiRxlBOvvEDLWe3uiSQxvSiRI1mCMUJG1DhfUsVozqZn&include=lineup,localteam,visi
torteam"

        res = requests.request("GET", url)
        if res.status_code == 200 and res:
            print(res.text)
            return json.loads(res.text)
```

```
  def contest(self,mid):
    url = "http://localhost:8080/contests/"+str(mid)
    res = requests.request("GET", url)
    print(res.json())
    if res.status_code == 200 and res:
      return (res.json())


  def quiz_gen(self):
    config = {
      "apiKey": "AIzaSyDiz4uJjcRpoklYOMQPshM6vMeY4Bmofzc",
      "authDomain":"sportsdapp-4fc33.firebaseapp.com" ,
      "databaseURL": "https://sportsdapp-4fc33.firebaseio.com",
      "storageBucket":"sportsdapp-4fc33.appspot.com",
      "serviceAccount": "sportsdapp.json"

    }
    firebase = pyrebase.initialize_app(config)
    db = firebase.database()
    u = "Questions/"

    #MONGO_URI = "mongodb+srv://Dhrupa:" + urllib.parse.quote("Dhrupa@337") +
"@cluster0.qfy6s.mongodb.net/test?retryWrites=true&w=majority"

    client = pymongo.MongoClient('localhost',27017)
    data = self.livematches()
    print(data)
    try:
      for k in data["data"]:
        mid = "2"+str(k['id'])
        print('mid', mid)

        qlink = list(client.CRICKETUPCOMINGLIVE[str(mid)].find({'_id': 'normal'}, {'_id': 0}))
        d2 = datetime.strptime(k['starting_at'], "%Y-%m-%dT%H:%M:%S.%fZ")
        d1 = datetime.now()+timedelta(hours = 24)
        if k['status'] == 'NS' and k['type'] not in set(['TEST','4day','Test/5day']) and d1>d2:
        # qlink = db.child("Questions/CRICKETUPCOMINGLIVE/"+mid).get()
          print('a')
          print(qlink)
          if (len(qlink) == 0 or '0' not in qlink[0]):

            if len(list(client.CRICKETUPCOMINGLIVE['scorecard'].find({'_id':str(mid)},{'_id':0})))==0:
              client.CRICKETUPCOMINGLIVE['scorecard'].insert_one({'_id':
str(mid),'status':k['status'],'League':k['league']['name'],'code':k['league']['name'],'local_team':k['localteam']['name'],'lo
cal_team_code':k['localteam']['code'],'visitor_team':k['visitorteam']['name'],'visitor_team_code':k['visitorteam']['code'
],'start_at':k['starting_at'],'round':k['round'],'local_flag':k['localteam']['image_path'],
'visitor_flag':k['visitorteam']['image_path'],'Batsman':None, 'Bowlers':None, 'Scorecard':None, 'result':'Result
Awaited', 'Type':k['type'], 'localteamsquad':None, 'visitorteamsquad':None})

            q_arr = self.Q_generation(mid, k,client)
            length = len(q_arr)
            contest = self.contest(mid)
            if len(list(client.CRICKETUPCOMINGLIVE['matches'].find({'_id':str(mid)},{'_id':0})))==0:
```

```
                client.CRICKETUPCOMINGLIVE['matches'].insert_one({'_id':str(mid),
'League':k['league']['name'],'code':k['league']['name'],'local_team':k['localteam']['name'],'local_team_code':k['localtea
m']['code'],'Type':k['type'],'visitor_team':k['visitorteam']['name'],'visitor_team_code':k['visitorteam']['code'],'start_at':
k['starting_at'],'round':k['round'],'local_flag':k['localteam']['image_path'],
'visitor_flag':k['visitorteam']['image_path'],'status':k['status'], 'contests':len(contest['contests']), 'questions':length})
                tcash = 0.0

                for n in range(0, length):
                    quiz = {}
                    quiz['reviewed'] = "0"
                    quiz["curparticipation"] = "1"
                    quiz['q_id'] = str(n)
                    quiz["match_id"] = mid
                    quiz["season_id"] = k["season_id"]
                    qqno = int(q_arr[n]['qno'])

                    if qqno < 3:
                        quiz['pfee'] = '3'
                        tcash += 1
                    elif qqno >= 3 and qqno <= 6:
                        quiz['pfee'] = '2'
                        tcash += 1
                    else:
                        quiz['pfee'] = '1'
                        tcash += 1

                    quiz['rewardsystem'] = '1'
                    quiz['creator'] = 'auto'
                    quiz['stopvoting'] = "0"
                    quiz['private'] = '0'
                    quiz['status'] = 'Question is Live.'
                    quiz['qno'] = q_arr[n]['qno']
                    quiz['teamid'] = q_arr[n]['teamid']
                    quiz['playerid'] = q_arr[n]['playerid']

                    today = date.today()
                    d2 = today.strftime("%B %d, %Y")
                # print("d2 =", d2)

                    quiz['desc'] = k["localteam"]["name"] + ' vs ' + k["visitorteam"]["name"] + ', ' + d2

                    if q_arr[n]["type"] == "1":
                        quiz["content"] = q_arr[n]['que']
                        quiz["type"] = "1"
                        quiz["options"] = q_arr[n]['options']
                        quiz["before"] = q_arr[n]['before']

                    else:
                        quiz["content"] = q_arr[n]['que']
                        quiz["options"] = []
                        quiz["type"] = "2"
                        quiz["before"] = q_arr[n]['before']

                    pfee = quiz['pfee']
```

```
                quiz = json.dumps(quiz)
                oplen = q_arr[n]['oplen']

                if len(list(client.CRICKET[str(mid)].find({'_id': str(mid)}, {'_id': 0}))) > 0:
                    client.CRICKET[str(mid)].update({'_id': str(mid)}, {'$set': {str(n):quiz}})
                else:
                    client.CRICKET[str(mid)].insert_one({'_id': str(mid), str(n): quiz})

                if len(list(client.CRICKETUPCOMINGLIVE[str(mid)].find({'_id': 'normal'}, {'_id': 0}))) > 0:
                    client.CRICKETUPCOMINGLIVE[str(mid)].update({'_id': 'normal'}, {'$set': {str(n): quiz}})
                else:
                    client.CRICKETUPCOMINGLIVE[str(mid)].insert_one({'_id': 'normal', str(n): quiz})
                # db.child(u).child("CRICKETUPCOMINGLIVE").child(mid).update({str(n):quiz})

                if len(list(client.CRICKETUPCOMINGLIVE['participants'].find({'_id': str(mid)}, {'_id': 0}))) > 0:
                    client.CRICKETUPCOMINGLIVE['participants'].update({'_id': str(mid)}, {'$set': {str(n): []}})
                else:
                    client.CRICKETUPCOMINGLIVE['participants'].insert_one({'_id': str(mid), str(n): []})

                opts = {'_id': str(mid)+str(n)}

                opno = 0
                while opno < oplen:
                    opts[opno]=0
                    opno += 1
                client.CRICKETUPCOMINGLIVE['participants'].insert_one(opts)

                # url='https://enigmatic-hamlet-61462.herokuapp.com/postQuestion'
                bcqid = str(mid) + str(n)
                print(bcqid)
                print(quiz)
                block.postQuestion(int(bcqid),q_arr[n]['que'],int(float(pfee)*10**18),pkey,pubkey)
                time.sleep(5)
                '''url = "http://localhost:8080/postQuestion"
                payload =
"qid={}&qcontent={}&pfee={}&options=6&type={}&pkey=0xfd32136c7a01c144494ca5bbd3779b77d1d6c09279d1af7c
2c5af9ed5d2d93b40db5&name=auto".format(bcqid, q_arr[n]['que'],pfee, q_arr[n]["type"])
                headers = {
                    'content-type': "application/x-www-form-urlencoded",
                    'cache-control': "no-cache",
                    'postman-token': "218578ec-41d6-1c3a-6135-462556660f35"
                }
                print(payload)
                response = requests.request("POST", url, data=payload, headers=headers)
                print(response.text)'''
                #db.child(u).child("CREATEDQCASH").child(mid).update({"CASH": str(tcash), "QUES": str(length)})
```

4. MAINTAINING OFFCHAIN DB

In the previous section, we understood that python scripts are continuously running to fetch the cricket and football data from Sportsmonk APIs. The raw data provided by Sportsmonk is transformed for our use cases and stored in offchain DB like MongoDB. It is done so as to reduce the calls on third party APIs, also to make the API calls on mobile application effective.

To start MongoDB service on Mac, run:

```
brew services start mongodb-community
```

Following variables are used to write into MongoDB collections:

```
usersdb = await client.db("Users").collection("users");
scorecardDB = await client.db("CRICKETUPCOMINGLIVE").collection("scorecard")
matchesDB = await client.db("CRICKETUPCOMINGLIVE").collection("matches");
```

- Collection users contains all the XIDs, public keys who participated in match ids and their statuses.

  ```
  users = client.CRICKETUPCOMINGLIVE.collection_names()
  quess = list(client.CRICKETUPCOMINGLIVE[str(j)].find({'_id':'test'},{'_id':0}))
  if j in set(['matches','scorecard','participants','qidWinnings']):
  ```

- Collection matches contain all the upcoming and live match ids that are relevant at the present time.

  ```
  client.CRICKETUPCOMINGLIVE['matches'].insert_one(
                          {'_id': str(mid), 'League': k['league']['name'], 'code': k['league']['name'],
  'local_team': k['localteam']['name'],'local_team_code':k['localteam']['code'], 'visitor_team':
  k['visitorteam']['name'], 'visitor_team_code':k['visitorteam']['code'], 'Type':'Test', 'start_at':
  k['starting_at'], 'round': k['round'], 'local_flag': k['localteam']['image_path'], 'visitor_flag':
  k['visitorteam']['image_path'], 'status':k['status'], 'contests':len(contest['contests']),
  'questions':0})
  ```

- Collection scorecard contains scorecard of all match ids after match is completed.

  ```
   client.CRICKETUPCOMINGLIVE['scorecard'].insert_one({'_id':
  str(mid),'status':k['status'],'League':k['league']['name'],'code':k['league']['name'],'local_team':
  k['localteam']['name'],'local_team_code':k['localteam']['code'],'visitor_team':k['visitorteam']['n
  ame'],'visitor_team_code':k['visitorteam']['code'],'start_at':k['starting_at'],'round':k['round'],'lo
  cal_flag':k['localteam']['image_path'],
  'visitor_flag':k['visitorteam']['image_path'],'Batsman':None, 'Bowlers':None,
  ```

'Scorecard':None, 'result':'Result Awaited', 'Type':'Test', 'localteamsquad':localSquad, 'visitorteamsquad':visitorSquad})

5. DATA & MODEL TRAINING FOR RESULTS PREDICTION

The data is available from howstat website for each player.
http://www.howstat.com/cricket/Statistics/Players/PlayerProgressSummary_ODI.asp?PlayerID=3474

The sample data looks like following:

Match	Innings	Date	Versus	Ground	Batting Runs	Batting Aggr	Batting Avg	Bowling Wkts	Bowling Aggr	Bowling Avg	Fielding Catches	Fielding Agg	Keeping Catches	Keeping Stumps	Keeping Agg	Age
1		23/06/2007	Ireland	Civil Service Cricket Club	-	0			0		1	1				20 years 54 days
2	1	26/06/2007	South Africa	Civil Service Cricket Club	8	8	8	0/3	0		1	2				20 years 57 days
3	2	05/10/2007	Australia	Rajiv Gandhi International Stadium	1	9	4.5		0			2				20 years 158 days
4	3	18/11/2007	Pakistan	Sawai Mansingh Stadium	52	61	20.33		0		1	3				20 years 202 days
5	4	03/02/2008	Australia	Brisbane Cricket Ground	29	90	22.5		0			3				20 years 279 days
6	5	04/02/2008	Sri Lanka	Brisbane Cricket Ground	0	90	18		0			3				20 years 280 days
7	6	10/02/2008	Australia	Melbourne Cricket Ground	39*	129	25.8		0		1	4				20 years 286 days
8	7	12/02/2008	Sri Lanka	Manuka Oval	70*	199	39.8		0			4				20 years 288 days

The training model is determined by following weights:

- Match No

- ❖ Innings No
- ❖ Date
- ❖ Versus
- ❖ Ground
- ❖ Age

The performance metrics are:

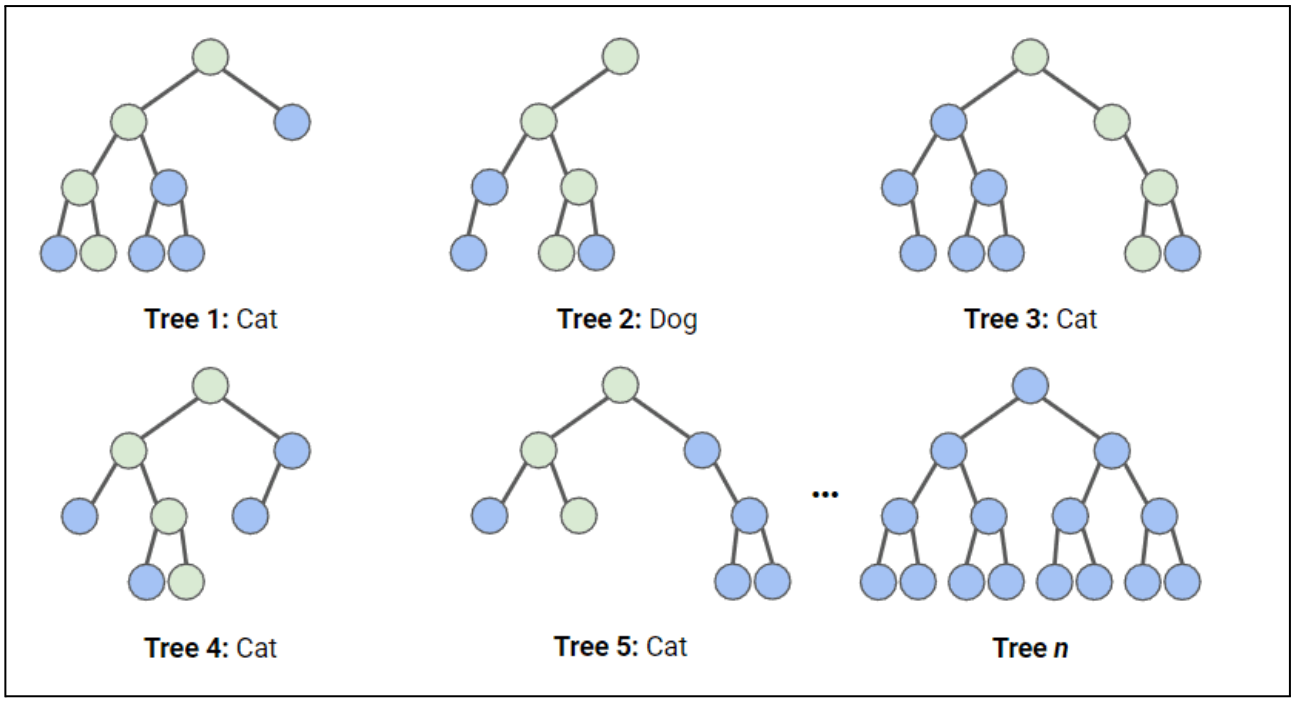
- ❖ Batting Runs
- ❖ Batting Aggr
- ❖ Batting Avg
- ❖ Bowling Wkts
- ❖ Bowling Aggr
- ❖ Bowling Avg
- ❖ Fielding Catches
- ❖ Fielding Agg
- ❖ Keeping Catches
- ❖ Keeping Stumps
- ❖ Keeping Agg

Random forests is a popular supervised machine learning algorithm that we used in order to train our model to predict the score of player in next match. Below are details of what random forest regression is:

- Random forests are for supervised machine learning, where there is a labeled target variable.
- Random forests are an ensemble method, meaning they combine predictions from other models.
- Random forests can be used for solving regression (numeric target variable) and classification (categorical target variable) problems.
- Each of the smaller models in the random forest ensemble is a decision tree.

How Random Forest works

Imagine you have a complex problem to solve, and you gather a group of experts from different fields to provide their input. Each expert provides their opinion based on their expertise and experience. Then, the experts would vote to arrive at a final decision.
In a random forest classification, multiple decision trees are created using different random subsets of the data and features. Each decision tree is like an expert, providing its opinion on how to classify the data. Predictions are made by calculating the prediction for each decision tree, then taking the most popular result. (For regression, predictions use an averaging technique instead.)
In the diagram below, we have a random forest with n decision trees, and we've shown the first 5, along with their predictions (either "Dog" or "Cat"). Each tree is exposed to a different number of features and a different sample of the original dataset, and as such, every tree can be different. Each tree makes a prediction. Looking at the first 5 trees, we can see that 4/5 predicted the sample was a Cat. The green circles indicate a hypothetical path the tree took to reach its decision. The random forest counts the number of predictions from decision trees for Cat and for Dog, and choose the most popular prediction.



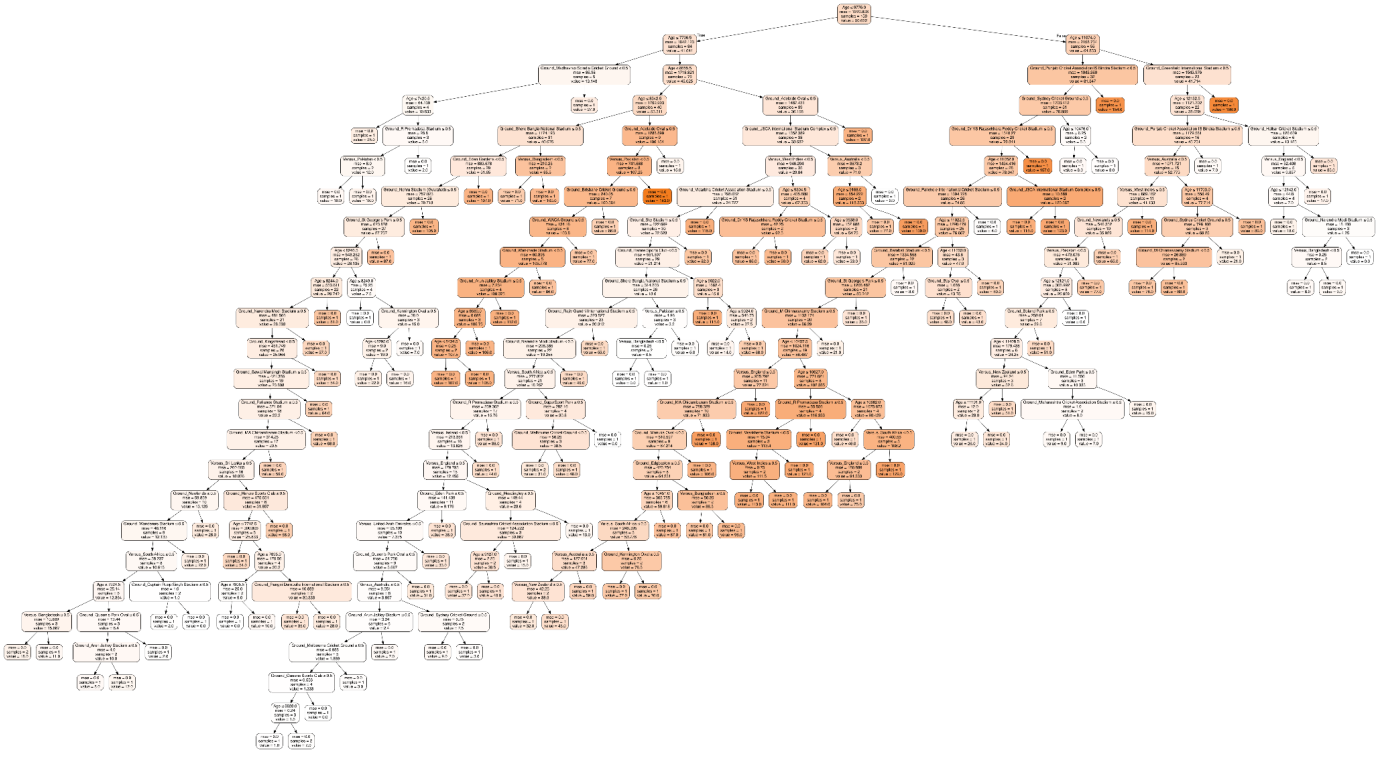
Python code to train model

The following code is used to train the model for data of players.

```
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_score, ConfusionMatrixDisplay

# Step 1: Data collection
# http://www.howstat.com/cricket/Statistics/Players/PlayerProgressSummary_ODI.asp?PlayerID=3474
data = pd.read_csv('player_data.csv')

# Step 2: Data preprocessing
X = data[['Versus', 'Ground', 'Age']] # features
y = data['Batting Runs'] # target variable
X = pd.get_dummies(X, columns=['Versus', 'Ground']) # one-hot encode categorical features
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=50) # split into training and test sets

# Step 3: Feature selection
# You can use techniques like correlation analysis, recursive feature elimination, or principal component analysis to
select relevant features

# Step 4: Model training
rf = RandomForestRegressor(n_estimators=100, random_state=0)
rf.fit(X_train, y_train)
```

```
# Step 5: Model evaluation
y_pred = rf.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)
print(f'Mean squared error: {mse:.2f}')
print(f'Root mean squared error: {rmse:.2f}')
print(f'R-squared: {r2:.2f}')

# accuracy = accuracy_score(y_test, y_pred)
# print("Accuracy:", accuracy)

# # # # Step 6: Prediction
next_match = pd.DataFrame({
   'Versus': ['Australia'],
   'Ground': ['Sydney Cricket Ground'],
   'Age': [12520]
})

# One-hot encode categorical features in next_match
next_match = pd.get_dummies(next_match, columns=['Versus', 'Ground'])
# Add missing columns to next_match (since not all values were present in X_train)
missing_cols = set(X_train.columns) - set(next_match.columns)
for col in missing_cols:
   next_match[col] = 0
next_match = next_match[X_train.columns] # Reorder columns to match X_train

# Predict batting score for next match
next_match_pred = rf.predict(next_match)
print(f'Predicted score for next match: {next_match_pred[0]:.2f}')
```

The decision tree for our use case came out to be like this:



6. DEVELOPMENT of APIs

In order to fetch data from MongoDB and use it in the application, we need to develop code with Node.js. Some of the major APIs are explained below:

1) http://localhost:8080/createAccount
   GET API
   It is to sign up and create the account for new user. Public key and private key are generated using Web3 modules.

2) http://localhost:8080/importAccount/:pk
   GET API
   It is to login and import the account for existing user who already have Public key and private key generated. Private key is needed to import the account.

3) http://localhost:8080/getxidsuggestions
   GET API
   It is to create XIDs suggestions for any new user combining simple names like Time, Past, Future, Dev, Fly etc.

4) http://localhost:8080/getXidSuggestionsByName/:name
   GET API
   It is to create XIDs suggestions for any new user using name provided by the user during signup.

5) http://localhost:8080/customQuestions
   POST API
   It is to create custom questions list by any user and create a private room in the app. The private room ID is generated for other users to join. This room ID is returned by the API.

6) http://localhost:8080/getQuestionsPrivate/:mid/:roomId
   GET API
   It is to get custom questions list by any user who has private room ID in the app. The questions selected by room owner and their details are returned by the API.

7) http://localhost:8080/getPastMatches
   GET API
   It is to get past matches' scorecards in which the user has participated.

8) http://localhost:8080/upcomingMatches
   GET API
   It is to get an upcoming list of cricket matches realtime from the CRICKETUPCOMINGLIVE MongoDB database.

9) http://localhost:8080/upcomingFootballMatches
   GET API

It is to get an upcoming list of football matches realtime from the FOOTBALLUPCOMINGLIVE MongoDB database.

10) http://localhost:8080/getQuestions/:game/:mid
    GET API
    It is to get a list of questions for a particular cricket match id for upcoming events in the match when the match is not started.

11) http://localhost:8080/getLiveQuestions/:mid
    GET API
    It is to get a list of questions for a particular cricket match id while the match is live and teams are playing.

12) http://localhost:8080/vote
    POST API
    It is to participate in questions of a match using question ID (generated by Match ID provided by sportsmonk APIs and question number). The participation is done directly on SportsFantasy smart contract.

13) http://localhost:8080/mymatches/:_game/:username
    GET API
    It is to get a list of match ids where the user participated in any question. Game can be cricket or football. Query is done using the username (XID) of user.

14) http://localhost:8080/getMyQuestions/:mid
    GET API
    It is to get list of question ids where the user participated in the queried match id.

15) http://localhost:8080/getWinningbalance/:public
    GET API
    It is to get the winning balance of any public key.

16) http://localhost:8080/submitsolution
    POST API
    It is called automatically after the match is finished to submit the solution so that result can be declared. Smart contract is directly called in this API.

17) http://localhost:8080/stopvoting
    POST API
    It is called automatically after the match is started so that noone can participate after match start or custom time based on question (like Toss). Smart contract is directly called in this API.

These APIs can be used in mobile applications. Just the network connection of the mobile should be the same as a PC running the Node.JS code and the server. We can also use the android/IOS emulator in the same PC for testing. After the testing is done, we can migrate localhost APIs to official domain APIs that can work on any network and the server codes in Node.JS to run on clouds like Digital Ocean, AWS etc.

7. FLUTTER APP DEVELOPMENT

Flutter is a mobile app development framework that uses the Dart programming language to build high-performance, cross-platform applications. In this case, Flutter has been used to implement the mobile application for user interaction and participation in upcoming and live match question events.

The development process starts with creating an account for users who are signing up for the first time. The public key and private key are generated, and a unique XID is chosen for the user that is linked to the public key. Flutter allows developers to easily create UI/UX components for the user sign-up process, including input forms for name, email, and password. The user's chosen XID can also be generated and stored in Flutter's local storage, allowing for seamless user authentication in future sessions.

Once the user has created an account and logged in, the home page is loaded. This page contains a list view of upcoming and live cricket and football matches. Flutter's widget system allows developers to easily create a list view that is scrollable, responsive, and dynamic, showing only the relevant matches to the user based on time and date.

When a user clicks on a specific match, the app navigates to a new screen where the questions for that match are loaded. In this screen, users can predict on any number of questions, choosing from multiple-choice or integer type options. Flutter allows developers to easily create dynamic user interfaces for these prediction screens, including forms and input widgets for capturing user input, as well as real-time updates on winning ratios.

Once the user has completed their predictions, the app navigates to a review screen, where the user can review their predicted options and confirm their participation fee. Flutter's plugin system allows developers to integrate payment gateways like Razorpay, which enable users to pay their participation fee using a variety of payment methods like credit/debit cards, net banking, and digital wallets.

After the user has paid their participation fee, the equivalent tokens are transferred to/from the user's public key, and the participation can be reviewed on the smart contract by anyone. Flutter's plugin system also allows developers to easily integrate blockchain technology into their apps, making it possible to implement transparent and fair transactions with timestamps.

Finally, once the match event is complete for a particular question, the reward is assigned to winners, and the balance of winnings can be viewed in the profile screen anytime. Flutter makes it easy to create dynamic, responsive UI/UX components for user profiles, including real-time updates on winnings and participation history.

Rationale

The rationale for mobile application UI/UX is listed below:

1. The first step is to create an account for users signing up for the first time in an application. The public key/private key is generated and XID is chosen for the user that is linked to the public key. The XID can be chosen by the user, it can be unique or something random. Suggestions for XID can be random or based on the user's name entered.
2. The home page is loaded after creating an account. In the homepage, upcoming & live matches of cricket and football are loaded as a list view.
3. Once an item in the list view, any cricket/football match is clicked, the questions for that match are loaded in a new screen.
4. In the next screen users can predict on any number of questions with MCQ or integer type. Once all questions the user wants to predict are selected, the review window containing predicted options and real time winning ratio is shown.
5. The user has to pay the total participation fee. If a user is participating for the first time, the balance must be 0. In order to add balance, we use Razorpay Plugin to receive or send the money. The equivalent tokens are transferred to/from the user's public key.
6. After the participation fee is deducted from the user and transferred to smart contract, participation can be reviewed on Smart contract by anyone, thereby keeping transactions transparent and fair with timestamps.
7. After the match event is complete for a particular question, the reward is assigned to winners and balance of winning can be viewed in the Profile screen anytime.

In conclusion, Flutter provides developers with a powerful and flexible framework for implementing mobile apps for user interaction and participation in upcoming and live match question events. With its rich set of UI/UX components, plugins, and integrations, Flutter makes it possible to create seamless and engaging user experiences that are optimized for performance and cross-platform compatibility.

8. CONCLUSION:

In conclusion, this dissertation presented the development of a blockchain-based sports prediction platform that uses smart contracts to facilitate quiz questions and automatically declare results after the match finishes. The platform is designed to be scalable, affordable, and transparent.

The platform is built using the Polygon network, which is known for its low transaction fees and fast transaction processing time. The smart contracts for the platform are coded in Solidity, and they are deployed on the Polygon testnet for testing purposes. The smart contracts were designed to meet the following requirements: quiz question generation, participation from different accounts, automatic result declaration after match finishes, winnings assignment for each quiz question, public retrieval of winnings, and user retrieval of winnings. The implementation of these requirements is expected to ensure that the platform is efficient and effective for users.

Moreover, the platform uses off-chain databases to store and manage data relevant to the platform. The off-chain databases are used to reduce the calls on third-party APIs and make API calls on mobile applications more effective. MongoDB is the database of choice for the platform.

The platform's development process involved the use of several tools and technologies, including Python, Flask, Web3.py, and Ganache. The development process was also iterative, involving several stages of testing and refinement to ensure that the platform meets the requirements and is user-friendly.

The platform's potential impact is significant, as it provides users with an opportunity to participate in sports quizzes and predict match outcomes, while also earning rewards for their accurate predictions. The platform's transparency ensures that users can trust the system, and its affordability makes it accessible to a wider range of users.

However, there are still some challenges that need to be addressed to ensure that the platform is fully functional and effective. For instance, the platform's reliance on Sportsmonk APIs for match data could pose a risk in cases where the APIs are not

available or provide inaccurate data. The platform also needs to address issues of data privacy and security to protect users' information and ensure that the platform is compliant with relevant regulations.

Overall, the blockchain-based sports prediction platform developed in this dissertation is a promising initiative that could revolutionize the sports prediction industry. The platform's innovative use of blockchain technology and off-chain databases provides users with a secure, scalable, and transparent platform for sports predictions. The potential for mass adoption of this platform is high, and it could open up new opportunities for users and stakeholders in the sports industry.

9. APPENDICES:

This report describes the implementation of a smart contract-based platform for creating quizzes based on cricket and football matches. The smart contract platform is developed using Polygon, an EVM compatible chain, and Solidity is used as the programming language for writing the smart contracts. The requirements for the smart contracts are as follows:

(a) Quiz questions generation

(b) Participation from different accounts (addresses)

(c) Declare result automatically after match finishes

(d) To assign the winnings for each quiz questions

(e) Fetch the winnings of any address publicly

(f) Retrieve the winnings by users

The smart contracts are designed to support two types of questions: integer and MCQ. The participants can join the quizzes by paying a participation fee, and the winnings are distributed among the users who predict the correct answer.

The implementation of the smart contract platform is supported by offchain databases such as MongoDB. The raw data from Sportsmonk APIs is transformed and stored in MongoDB collections to make API calls on the mobile application effective. The collections contain information about users who participated in match ids and their statuses and upcoming and live match ids that are relevant at the present time.

The implementation of the smart contract-based platform is intended to be scalable, cost-effective, and transparent. Future work may include creating an oracle for sources of truth from sports APIs, which can be run by nodes independently to further enhance the transparency and reliability of the platform.

10. LITERATURE REFERENCES:

The following are referred journals from the preliminary literature review.

[1] A. Gervais, G. O. Karame, K. Wust, V. Glykantzis, H. Ritzdorf and S. Capkun: "On the security and performance of proof of work blockchains". ACM SIGSACconference (2016).

[2] iI. C. Lin and T. C. Liao: "A Survey of Blockchain Security Issues and Challenges". IJ Network Security, 19(5), pp.653-659 (2017).

[3] LeCun, Y., Bengio, Y. & Hinton, G. Deep learning. Nature 521, 436–444 (2015).

[4] Types of Fantasy Sports Users and Their Motivations Lee K. Farquhar, Robert Meeds Journal of Computer-Mediated Communication, Volume 12, Issue 4, 1 July 2007, Pages 1208–1228

[5] Andrew C. Billings & Brody J. Ruihley (2013) Why We Watch, Why We Play: The Relationship Between Fantasy Sport and Fanship Motivations, Mass Communication and Society, 16:1, 5-25, DOI: 10.1080/15205436.2011.635260

11. GLOSSARY

- Blockchain Smart Contracts:
    - Blockchain: a decentralized digital ledger used to store data securely.
    - Smart contracts: self-executing contracts with the terms of the agreement directly written into code, stored on a blockchain.
    - Source of truth: a reliable, authoritative source of information used as a reference for data.
    - Chains: a blockchain network or platform.
    - Polygon: a Layer 2 scaling solution for Ethereum that provides faster and cheaper transactions.
    - Proof of Stake (PoS): a consensus mechanism for blockchains that uses validators to validate transactions and create new blocks, and their influence is based on the amount of cryptocurrency they hold.
    - Solidity: a programming language used to write smart contracts for the Ethereum blockchain.
    - Testnet: a blockchain network used for testing smart contracts and applications.
    - Quiz questions generation: a function that creates match questions on-chain, generated by a python script.
    - Addresses: unique identifiers that represent an account on a blockchain.
    - IPFS: InterPlanetary File System, a protocol and network designed to create a content-addressable, peer-to-peer method of storing and sharing hypermedia in a distributed file system.
    - Participation fee: the fee paid by users to participate in the quiz.
    - MCQ: multiple-choice questions.

- Maintaining Offchain DB:
    - Python: a programming language used to write scripts for fetching data from SportsMonk APIs.
    - SportsMonk: an API service that provides real-time sports data for various sports.
    - Raw data: the data as it is provided by SportsMonk APIs.
    - Transformed data: data that is processed and prepared for use in the blockchain application.
    - Offchain DB: a database that is not stored on a blockchain, used for storing data that does not require the security and immutability of a blockchain.
    - MongoDB: a popular NoSQL document-based database.

- Collection: a grouping of MongoDB documents that are stored together.
- XIDs: a pseudo name for the question creator used to secure Personally Identifiable Information (PII).
- Public keys: a unique identifier used to verify ownership of an account on a blockchain.
- Statuses: the current state of an account or transaction on a blockchain.
- Upcoming match IDs: IDs of upcoming matches that are relevant at the present time.
- Live match IDs: IDs of live matches that are relevant at the present time.

Checklist of Items for the Final Dissertation

This checklist is to be duly completed, verified and signed by the student.

1.	**Is the final report neatly formatted with all the elements required for a technical Report?**	✓ Yes/ No
2.	Is the Cover page in proper format as given in Annexure A?	✓ Yes/ No
3.	Is the Title page (Inner cover page) in proper format?	✓ Yes/ No
4.	(a) Is the Certificate from the Supervisor in proper format? (b) Has it been signed by the Supervisor?	✓ Yes/ No ✓ Yes/ No
5.	Is the Abstract included in the report properly written within one page? Have the technical keywords been specified properly?	✓ Yes/ No
6.	Is the title of your report appropriate? **The title should be adequately descriptive, precise and must reflect scope of the actual work done.** Uncommon abbreviations / Acronyms should not be used in the title	✓ Yes/ No
7.	Have you included the List of abbreviations / Acronyms?	✓ Yes/ No
8.	Does the Report contain a summary of the literature survey?	✓ Yes/ No
9.	Does the Table of Contents include page numbers? (i). Are the Pages numbered properly? (Ch. 1 should start on Page # 1) (ii). Are the Figures numbered properly? (Figure Numbers and Figure Titles should be at the bottom of the figures) (iii). Are the Tables numbered properly? (Table Numbers and Table Titles should be at the top of the tables) (iv). Are the Captions for the Figures and Tables proper? (v). Are the Appendices numbered properly? Are their titles appropriate	✓ Yes/ No
10.	Is the conclusion of the Report based on discussion of the work?	✓ Yes/ No
11.	Are References or Bibliography given at the end of the Report? Have the References been cited properly inside the text of the Report? Are all the references cited in the body of the report	✓ Yes/ No
12.	Is the report format and content according to the guidelines? The report should not be a mere printout of a PowerPoint Presentation, or a user manual. Source code of software need not be included in the report.	✓ Yes/ No

Declaration by Student:
I certify that I have properly verified all the items in this checklist and ensure that the report is in proper format as specified in the course handout.

Jaspreet Singh

Place: _____Delhi_____ **Signature of the Student**

Date: _____17/04/2023___ **Name: ___Jaspreet Singh_____**

ID No.: ___2021MT93340_____