Fantasy English Premier League Players Marketplace

Name	UBID	Email
Sivakumar Pasupathi	50366350	spasupat@buffalo.edu
Sai Prashanth Selvaganapathy	50419614	saiprash@buffalo.edu

Project Objective:

The marketplace will enable users/teams in the English Premier League to buy and sell players.

Issues addressed:

- *Secure Environment:* Ability to register and track players sale and purchase list without any tampering
- *Transparency:* ability to check and verify each transaction by verifying the ledger
- *Smart Contract:* Able to perform advance transactions like swapping players for a fee/difference in cost.
- *Trade*: Ability to sell players for a price higher than initial cost

Abstract:

To create a marketplace where anyone and everyone can buy and sell English premier league players with ease. To achieve this a single market place is created where users can sell and buy players. As with any marketplace, trust is an essential part, for this the blockchain protocol plays an important role. All transactions will be registered in a distributed ledger; this will be immutable so that no fraud can occur. A new coin will be issued for the transaction. This marketplace will provide convenience and transparency for any future leagues to use.

Major Functionalities :

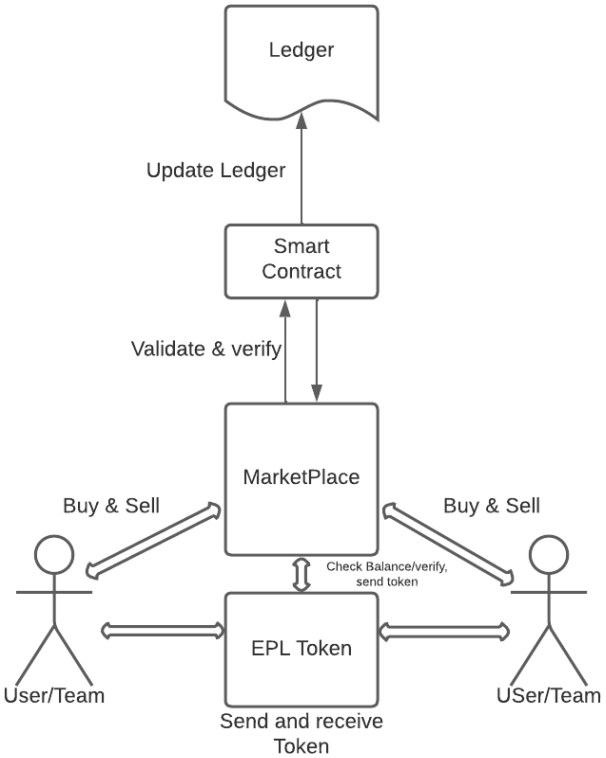
- Buy Players
- Update Sellable(Sell players option)
- Set Player Price
- Add/Update Team
- Add Player

Token Symbol:

The logo is a coin/currency with a football in the centre. The football symbolises the English Premier League which is the bases of the marketplace

Coin Name: Fantasy EPL Coin
MarketPlace Name: Fantasy EPL

UML Use case diagram:



UI/Wire frame:

Contract Diagram:

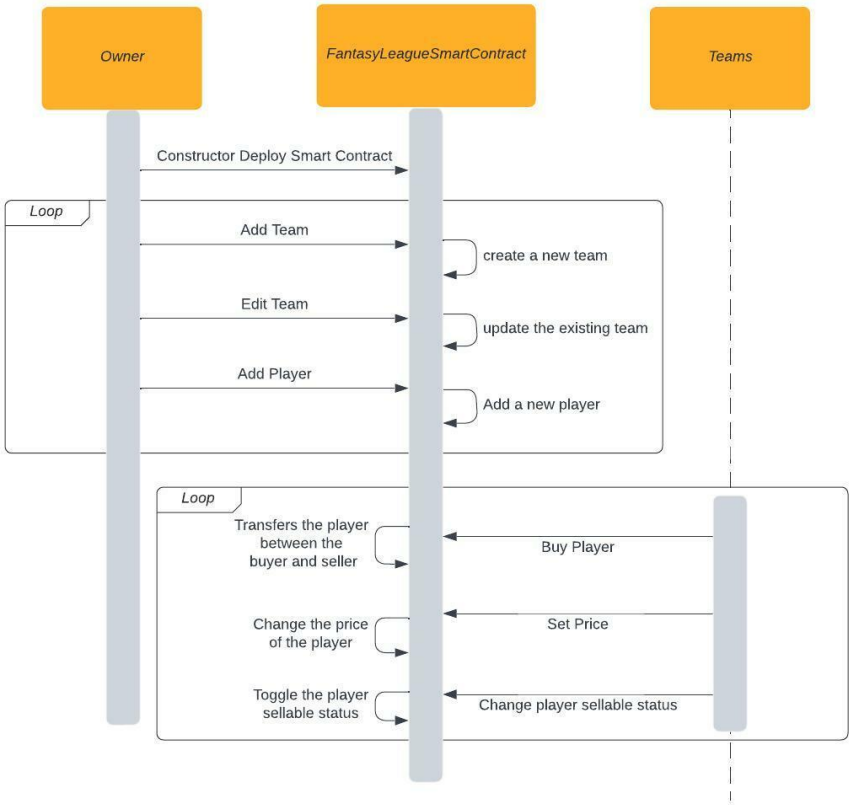
FantasyLeague
address **owner** *Struct* **Teams** *uint* **teamsCount** *Mapping* **teams** *struct* **Player** *uint* **playersCount** *Mapping* **players**
modifier onlyOwner *checkIfTeamRegistered()* *checkIfPlayerIsInTeam()* *checkIfPlayerExist()* *checkIfPlayerSellable()* *checkifPlayerAlreadyOwned()* *checkMyPlayerIndex()*
addTeam() *addPlayer()* *setPlayerPrice()* *togglePlayerSellable()* *buyPlayer()* *getMyPlayersCount()* *getMyPlayer()* *getAllPlayersCount()* *getPlayer()* *checkIfTeamExist()* *getMyTeamDetails()* *getTokenBalance()*

FantasyEPLToken
address **`allowedMarketPlace`** *Address* ***`creator`***
modifier `onlyMarketPlace` modifier onlyOwner
approve() *setAllowedMarketPlace()* *getAllowedMarketPlace()*

Quad Chart:

Use Case: Fifa Marketplace To create a decentralized application where Fifa teams can buy, sell players, update player cost wiith ease and without any fraud.	**Current Issue:** In the current model of a centralized server, any unaotherized person may gain access, edit players, cost etc.
Blockchain based solution: A smart contract is deployed by the owner of the league He has the authority to add teams and players of his choice Each team can buy and sell players that are available	**Benefits:** By saving all the transaction that takes place on a distributed ledger, every action is immutable, this provides a level of security that a centralized serve can't do.

Sequence Diagram:



ERC-20 Functionalities:

The ERC-20 Standard is used to create tokens on the Ethereum Blockchain. The OpenZeppelin library is used for creating the ERC-20 token.

_mint:
When the ERC-20 token is deployed, we are minting 10000 tokens and assigning it to the deployer(owner).

balanceOf:
Used to check the token balance of the address.

approve:
This method approves a user to transfer tokens on behalf of the user. Approval is done by updating the allowance. In FantasyEPLToken(ERC-20 Token) we have overridden this method to make the marketplace auto-approved for transferring tokens.

transferFrom:
Using this method, tokens can be transferred from one account to another account. The address calling this method should have been approved to transfer the tokens. So the ‘approve’ method is called before the ‘transferFrom’ method.

ERC-20 Token Interaction with the MarketPlace Contract:

allowedMarketPlace:
Inside the ERC-20 Token we set allowedMarketPlace. This is the address of the MarketPlace which is allowed to transfer tokens on behalf of the user. This operation can be performed only by the deployer.

Creating Token Instance inside MarketPlace:
When the MarketPlace contract is deployed, we pass the contract address of the Token. Using this, a token instance is created inside the MarketPlace. We call various token functionalities using this instance.
Functionalities of Token called from the MarketPlace are:

- balanceOf
- approve
- transferFrom

Contract Data Structure:

Player Struct:

```
struct Player{
        uint id;
        string name;
        uint price;
        bool is_sellable;
        address payable ownedBy;
    }
```

The above struct contains info about players such as name, id, price, is_sellable.
An additional information called ownedBy is used to indicate the current owner address of the player and transfer the money is sold to the owner of the player

Team Struct:

```
struct Teams{
        uint id;
        string name;
        uint[] ownedPlayers;
    }
```

Contains info about team and the players they own in an array of IDs

Team mapping:

```
//List of Participating Teams
    mapping(address => Teams) public teams;
```

Contains the address mapping of each individual team.

Player Mapping:

```
mapping(uint => Player) players;
```

Mapping of players struct with ID

OnlyOwner modifier:

```
modifier onlyOwner(){
        require(msg.sender == owner);
        _;
    }
```

Add Team:

```
function addTeam(address teamAddress, string memory _name) public
onlyOwner{
        uint[] memory ownedPlayers;
        teams[teamAddress] = Teams(teamsCount+1, _name, ownedPlayers);
        teamsCount = teamsCount + 1;
    }
```

Option to add team by the contract owner

Add Player:

```
function addPlayer(string memory name, uint price, bool is_sellable)
public onlyOwner{
        players[playersCount] = Player(playersCount, name, price,
is_sellable, owner);
        assignPlayersToTeam(owner, playersCount);
        playersCount = playersCount + 1;
    }
```

Add player by the owner of the contract

Buy Player:

```
function buyPlayer(uint playerID) public payable
```

Checks if a player Id,amount and team is valid. If so buys the player from seller and transfers the amount.

Get Player count:

```
function getAllPlayersCount() public view returns(string memory){
        return toString(playersCount);
    }
```

Get total player count, used in UI

Update Sellable:

```
function togglePlayerSellable(uint playerID) public{
        checkIfPlayerIsInTeam(msg.sender, playerID);
        players[playerID].is_sellable = !players[playerID].is_sellable;
    }
```

Marks or unmarked a player as sellable

Set Player price:

```
function setPlayerPrice(uint playerID, uint price) public{
        checkIfPlayerIsInTeam(msg.sender, playerID);
        players[playerID].price = price;
    }
```

Update player price if the user is the owner of the player

Get Player Info:

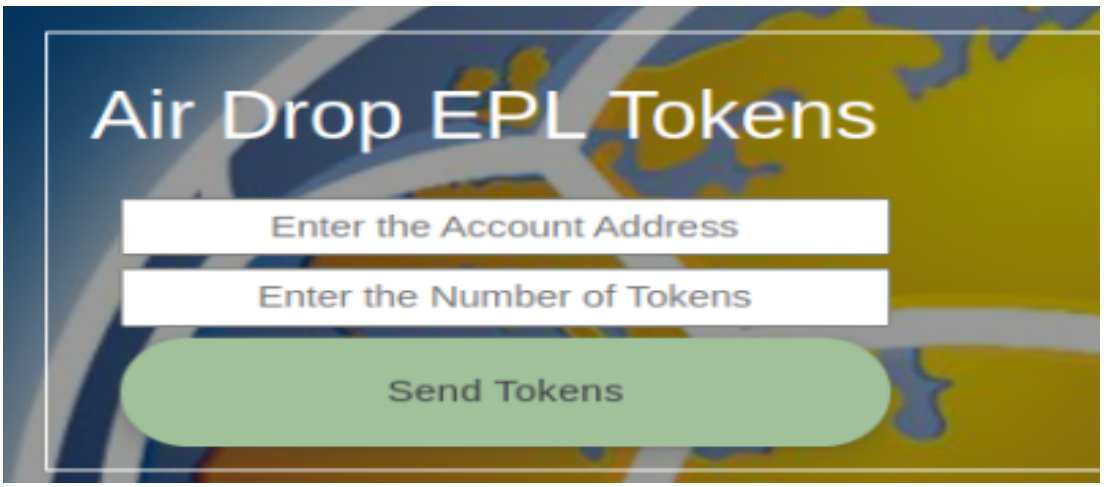
```
function getPlayer(uint index) public view returns(string memory,
string memory, string memory, bool, address){
        Player memory player = players[index];
        return (toString(player.id), player.name,
toString(player.price), player.is_sellable, player.ownedBy);
    }
```

Get the info of the player to display in the UI

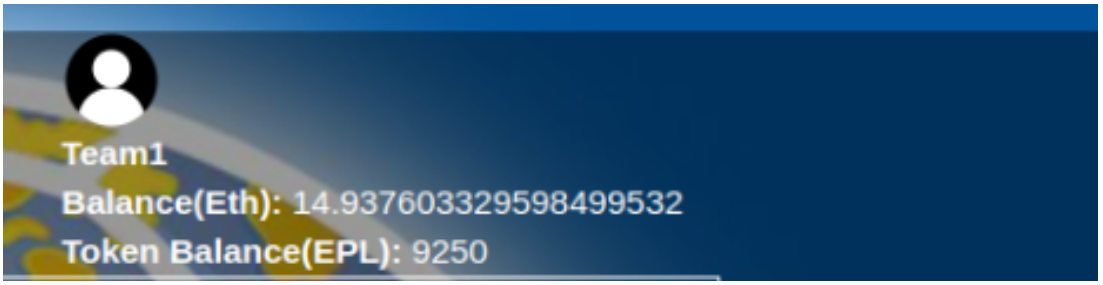
Functionality:

Buy Players: Ability to buy players that are available, plays that are marked sellable by owners can other team buy them.

AirDrop: The Owner of the contract can send EPL Token to any other team/account

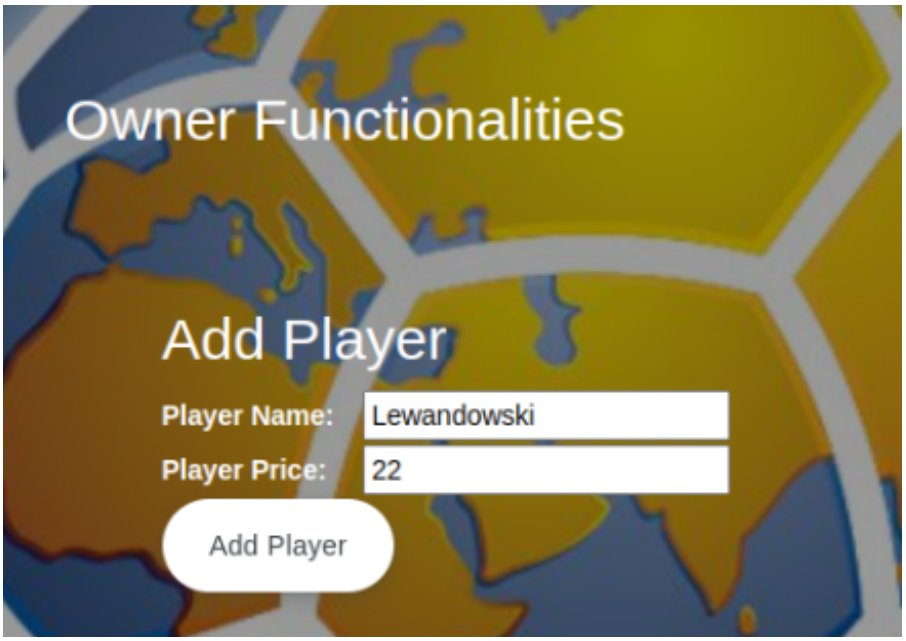


Balance Token view: View the balance token and Ethers(For gas fees) in the UI

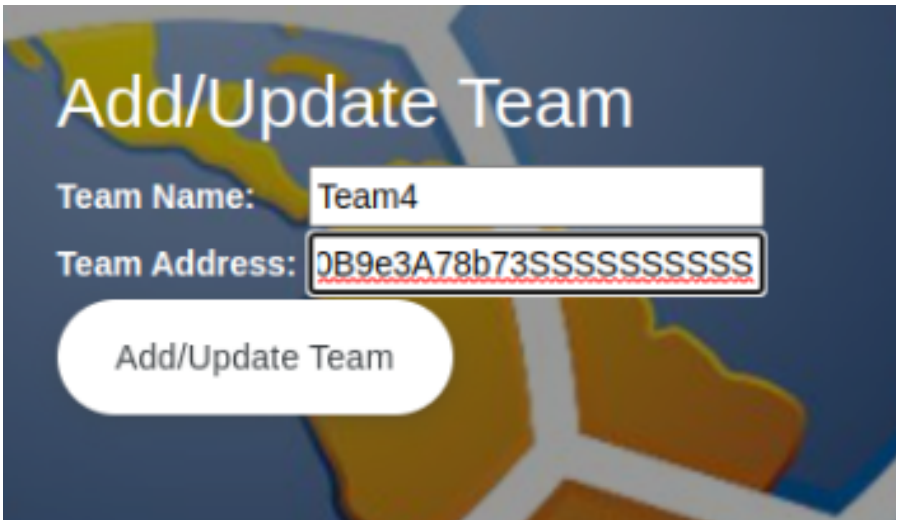


Trade(Update cost): Owners can buy players and update their price to sell them at higher cost if they choose to do so.

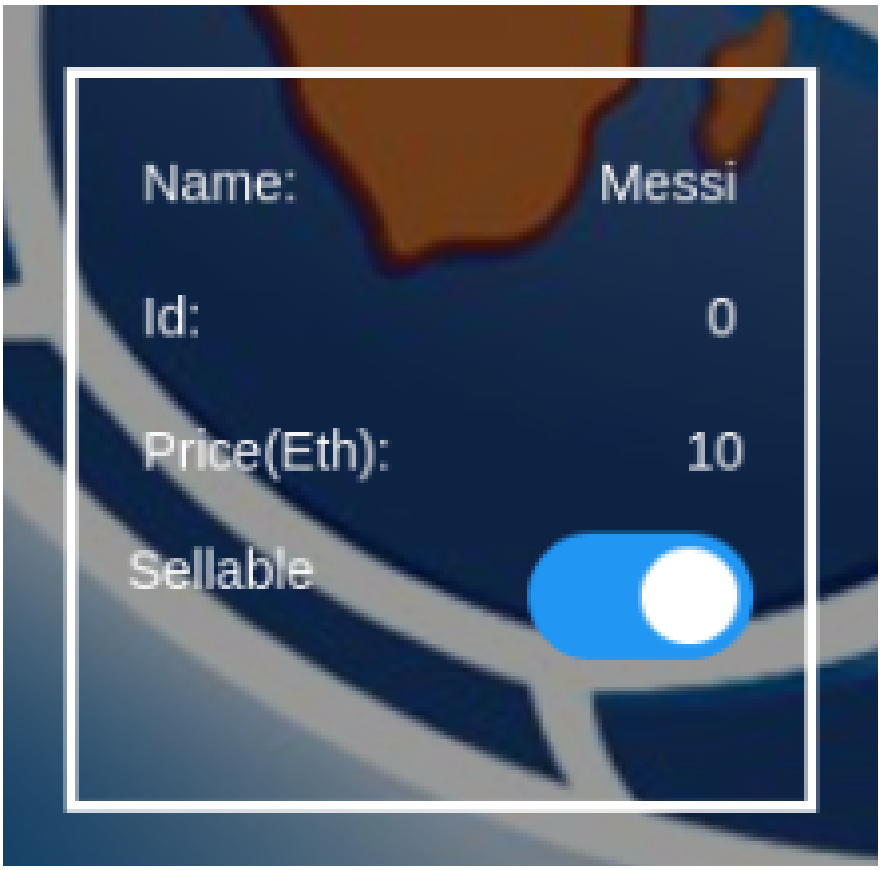
Add Player: Owner of the league can add players if and when needed



Add Team: Owner of the league can add/register new teams.



Update Sellable: Owner of a player can choose whether or not to have the player sellable.

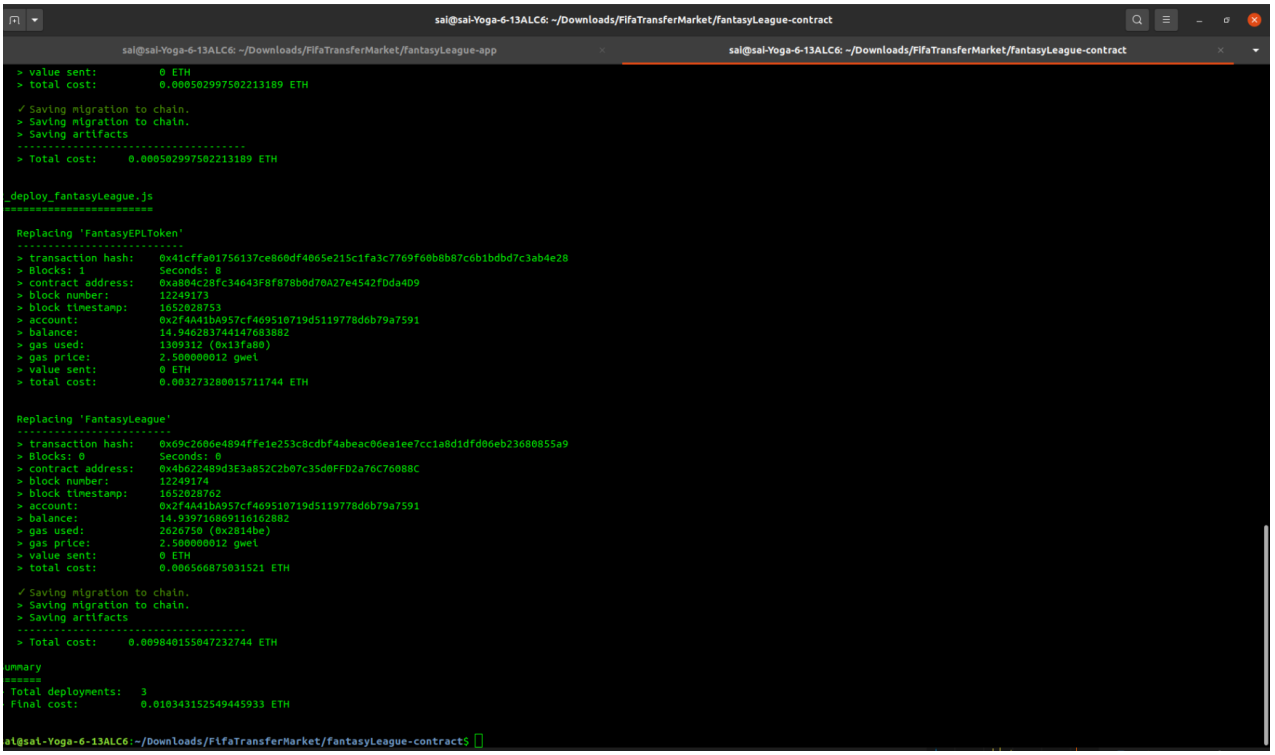


UI Owner Prespective:

Instruction to Deploy

- Inside the fantasyLeague-contract folder, open the .env file and update the MNEMONIC to the account of your choice (contract will be deployed via this account)
- Run the command **'truffle migrate –reset –network ropsten'** , this will deploy the contract on the test network.
- Now, inside the fantasyLeague-app folder, run the command 'npm start', this will launch the app on port 3000 i.e localhost:3000 (Use npm install if node module is not available)

Note Our deployed contract address are:
FantasyEPLToken: 0xa804c28fc34643F8f878b0d70A27e4542fDda4D9
FantasyLeague: 0x4b622489d3E3a852C2b07c35d0FFD2a76C76088C



Solidity Code:

FantasyLeague.sol

```
pragma solidity ^0.8.0;

import "./FantasyEPLToken.sol";

contract FantasyLeague{
   address payable owner;

   FantasyEPLToken token;
```

```
    constructor(address _token){
        owner = payable(msg.sender);
        addTeam(owner, 'Team1');
        token = FantasyEPLToken(_token);
    }

    //Only Owner
    modifier onlyOwner(){
        require(msg.sender == owner);
        _;
    }

    function getOwner() public view returns(address){
       return owner;
    }


    function getAllowedMarketPlace() public view returns(address){
        return token.getAllowedMarketPlace();
    }

    function getTokenBalance() public view returns(string memory){
        return toString(token.balanceOf(msg.sender));
    }

    //Strcuture for Teams
    //Each User who are authorized to buy and sell is a Team
    struct Teams{
        uint id;
        string name;
        uint[] ownedPlayers;
    }

    uint teamsCount = 0;
    //List of Participating Teams
    mapping(address => Teams) public teams;

    function addTeam(address teamAddress, string memory _name) public
onlyOwner{
        uint[] memory ownedPlayers;
        teams[teamAddress] = Teams(teamsCount+1, _name, ownedPlayers);
        teamsCount = teamsCount + 1;
    }
```

```
    function assignPlayersToTeam(address payable teamAddress, uint
playerID) public onlyOwner{
        checkIfTeamRegistered(teamAddress);
        teams[teamAddress].ownedPlayers.push(playerID);
        players[playerID].ownedBy = teamAddress;
    }

    struct Player{
        uint id;
        string name;
        uint price;
        bool is_sellable;
        address payable ownedBy;
    }

    uint public playersCount = 0;

    mapping(uint => Player) players;

    function checkIfTeamRegistered(address teamAddress) internal view{
        if(teams[teamAddress].id == 0){
            revert("Team Not Registered - Contact Owner");
        }
    }

    function checkIfPlayerIsInTeam(address teamAddress, uint playerID)
internal view{
        checkIfTeamRegistered(teamAddress);
        checkIfPlayerExist(playerID);
        Teams memory team = teams[teamAddress];

        bool playerPresentInTeam = false;
        for(uint i = 0; i < team.ownedPlayers.length; i++){
            if(team.ownedPlayers[i] == playerID){
                playerPresentInTeam = true;
            }
        }

        if(!playerPresentInTeam){
            revert("Player is not Owned by the Team");
        }
```

```
    }

    function checkIfPlayerExist(uint playerID) internal view{
        if(playerID >= playersCount){
            revert("Player Not Present");
        }
    }

    function addPlayer(string memory name, uint price, bool is_sellable)
public onlyOwner{
        players[playersCount] = Player(playersCount, name, price,
is_sellable, owner);
        assignPlayersToTeam(owner, playersCount);
        playersCount = playersCount + 1;
    }

    function setPlayerPrice(uint playerID, uint price) public{
        checkIfPlayerIsInTeam(msg.sender, playerID);
        players[playerID].price = price;
    }

    function togglePlayerSellable(uint playerID) public{
        checkIfPlayerIsInTeam(msg.sender, playerID);
        players[playerID].is_sellable = !players[playerID].is_sellable;
    }

    function checkIfPlayerSellable(uint playerID) internal view
returns(bool){
        checkIfPlayerExist(playerID);
        if(!players[playerID].is_sellable){
            revert("Player Not Sellable");
        }
        return players[playerID].is_sellable;
    }

    function checkTokenBeforePurchase(uint amount) internal view{
        if(amount > token.balanceOf(msg.sender)){
            revert("Not Enough Tokens");
        }
    }

    function sendToken(address to, uint amount) public{
        token.approve(msg.sender,amount);
```

```
        token.transferFrom(msg.sender, to, amount);
    }

    function buyPlayer(uint playerID, uint amount) public payable{
        checkIfPlayerExist(playerID);
        checkifPlayerAlreadyOwned(playerID);
        //checkIfPlayerIsInTeam(msg.sender, playerID);
        checkIfPlayerSellable(playerID);

        Player memory player = players[playerID];

        address payable seller = player.ownedBy;
        // uint amount;
        // amount = msg.value;
        uint price;
        price = player.price;
        if (amount != price){
            revert("Mismatch price");
        }

        checkTokenBeforePurchase(amount);

        sendToken(seller, amount);
        //seller.transfer(msg.value);

        if(seller != owner){
            //owner.transfer(price/10);
            removePlayerFromSeller(playerID, seller);
        }

        addPlayerToSelf(playerID);


    }
    function checkifPlayerAlreadyOwned(uint playerID) internal view{
        Player storage player = players[playerID];
        if (player.ownedBy == (msg.sender)){
            revert("Already owned");
        }
    }
    function addPlayerToSelf(uint playerID) internal {
        Player storage player = players[playerID];
        player.ownedBy = payable(msg.sender);
```

```
        teams[msg.sender].ownedPlayers.push(playerID);
    }

    function removePlayerFromSeller(uint playerID, address payable
seller) internal {

        uint[] memory ownedPlayers = teams[seller].ownedPlayers;
        uint[] memory updatedOwnedPlayers;
        teams[seller].ownedPlayers = updatedOwnedPlayers;

        for(uint i = 0; i < ownedPlayers.length; i++){
            if(ownedPlayers[i] != playerID){
                teams[seller].ownedPlayers.push(ownedPlayers[i]);
            }
        }

    }

    function getMyPlayersCount() public view returns(string memory){
        return toString(teams[msg.sender].ownedPlayers.length);
    }

    function checkMyPlayerIndex(uint index) internal view returns(bool){
        checkIfTeamRegistered(msg.sender);
        if(index >= teams[msg.sender].ownedPlayers.length){
            return false;
        }
        return true;
    }

    function getMyPlayer(uint index) public view returns(string memory,
string memory, string memory, bool){
        checkIfTeamRegistered(msg.sender);
        require(checkMyPlayerIndex(index), "Searching Player not
Available");

        Player memory player =
players[teams[msg.sender].ownedPlayers[index]];
        return (toString(player.id), player.name,
toString(player.price), player.is_sellable);
    }

    function getAllPlayersCount() public view returns(string memory){
```

```
        return toString(playersCount);
    }

    function getPlayer(uint index) public view returns(string memory,
string memory, string memory, bool, address){
        Player memory player = players[index];
        return (toString(player.id), player.name,
toString(player.price), player.is_sellable, player.ownedBy);
    }

    function checkIfTeamExist() external view returns(bool){
        if(teams[msg.sender].id == 0){
            return false;
        }
        return true;
    }

    function getMyTeamDetails() public view returns(uint, string
memory){
        checkIfTeamRegistered(msg.sender);
        Teams memory myTeam = teams[msg.sender];
        return (myTeam.id, myTeam.name);
    }

     function toString(uint256 value) internal pure returns (string
memory) {
        //
https://github.com/oraclize/ethereum-api/blob/b42146b063c7d6ee1358846c1
98246239e9360e8/oraclizeAPI_0.4.25.sol

        if (value == 0) {
            return "0";
        }
        uint256 temp = value;
        uint256 digits;
        while (temp != 0) {
            digits++;
            temp /= 10;
        }
        bytes memory buffer = new bytes(digits);
        while (value != 0) {
            digits -= 1;
            buffer[digits] = bytes1(uint8(48 + uint256(value % 10)));
```

```
            value /= 10;
        }
        return string(buffer);
    }
}
```

Token:

```
pragma solidity ^0.8.0;

import "openzeppelin-solidity/contracts/token/ERC20/ERC20.sol";

contract FantasyEPLToken is ERC20{

    address creator;

    address allowedMarketPlace;

    constructor(string memory name_, string memory symbol_, uint supply)
ERC20(name_, symbol_) {
        creator = msg.sender;
        _mint(msg.sender, supply * (10 ** decimals()));
    }

    //Only MarketPlace
    modifier onlyMarketPlace(){
        require(msg.sender == allowedMarketPlace);
        _;
    }

    //Only Owner
    modifier onlyOwner(){
        require(msg.sender == creator);
        _;
    }

    function approve(address spender, uint256 amount) public virtual
override returns (bool) {
        //address owner = _msgSender();
        _approve(spender, allowedMarketPlace, amount);
        return true;
    }
```

```
   function setAllowedMarketPlace(address _allowedMarketPlace) public
onlyOwner{
        allowedMarketPlace = _allowedMarketPlace;
   }

   function getAllowedMarketPlace() public view onlyMarketPlace
returns(address){
        return allowedMarketPlace;
   }

}
```

Web3 UI code (app.js):

```
App = {
 web3Provider: null,
 contracts: {},
 account: '0x0',
 Owner: '0x0',
 fantasyLeagueInstance:null,
 //fantasyEPLTokenInstance:null,

 init: function() {
   return App.initWeb3();
 },

 initWeb3: function() {
   if (typeof web3 !== 'undefined') {
     App.web3Provider = web3.currentProvider;
     web3 = new Web3(web3.currentProvider);
   }
   web3.eth.defaultAccount=web3.eth.coinbase;

   window.ethereum.on('accountsChanged', function (accounts) {
     location.reload();
   })
   return App.initContract();
 },

 initContract: function() {
```

```
  $.getJSON("FantasyLeague.json", function(fantasyLeague) {

    window.ethereum.enable();
    App.contracts.FantasyLeague = TruffleContract(fantasyLeague);
    App.contracts.FantasyLeague.setProvider(App.web3Provider);


    App.contracts.FantasyLeague.deployed().then(function(instance) {
      App.fantasyLeagueInstance = instance;
      return App.fantasyLeagueInstance.getOwner();
    }).then(function(owner) {
      App.Owner = owner;
      App.checkIfTeamRegistered();
    }).catch(function(error) {
     console.warn(error);
    });
  });

  web3.eth.getCoinbase(function(err, account) {
    console.log(web3.eth);
    console.log("Account", account);
    if (err === null) {
      App.account = account;
      $("#accountAddress").html("Your Account: " + account);
    }
  });
 },
  loadPlayers   : function(){

App.fantasyLeagueInstance.getAllPlayersCount().then(function(playersCou
nt){
    for(let i = 0; i < playersCount; i++){
      App.fantasyLeagueInstance.getPlayer(i).then(renderPlayer);
    }
  });
 },

 loadTeamDetails : function(){
  App.fantasyLeagueInstance.getMyTeamDetails().then(setTeamDetails);
  App.fantasyLeagueInstance.getTokenBalance().then(setTokenBalance);
  setAccountBalance();
 },
```

```
 renderOwnerFunctionalities : function(){
   jQuery("#owner-div").removeClass("hidden");
 },

 checkIfTeamRegistered : function(){

App.fantasyLeagueInstance.checkIfTeamExist().then(function(isTeamRegist
ered){
     if(isTeamRegistered){
       App.renderPage();
     }
     else{
       // alert("Team Not Registered");
     }
   });
 },

 renderPage: function() {
   if(App.Owner == App.account){
     App.renderOwnerFunctionalities();
     document.querySelector("#air-drop-container").style.visibility =
"visible";
   }else{
     node = document.querySelector("#air-drop-container");
     node.parentElement.removeChild(node);
   }
   App.loadPlayers();
   App.loadTeamDetails();
 },

 addPlayer : function(){
   let playerName = jQuery('#player-name').val();
   let playerPrice = web3.toWei(jQuery('#player-price').val(),
"ether");
   App.fantasyLeagueInstance.addPlayer(playerName, playerPrice,
true).then(function(){
     alert("Player Added Successfully");``
     location.reload();
   });
 },

 addTeam : function(){
   let teamName = jQuery('#team-name').val();
```

```
    let teamAddress = jQuery('#team-address').val();
    App.fantasyLeagueInstance.addTeam(teamAddress,
teamName).then(function(){
      alert("Team Added Successfully");
    });
  },

  buyPlayer : function(id, price){
    App.contracts.FantasyLeague.deployed().then(function(instance){
      return instance.buyPlayer(id, web3.toWei(price, "ether"), {gas:
3000000});
    }).then(function(result){
      console.log(result);
      location.reload();
    }).catch(function(error) {
      console.warn(error);
      location.reload();
    });
  },
  changeSellable : function(id){
    App.contracts.FantasyLeague.deployed().then(function(instance){
      return instance.togglePlayerSellable(id, {gas: 3000000});
    }).then(function(result){
      console.log(result);
      location.reload();
    }).catch(function(error) {
      console.warn(error);
    });
  },

  updatePrice : function(id, event){
    console.log("aaaaaaaa", id, event.keyCode  );
    if (event.keyCode !=13){
      return;
    }
    //console.log(event)

    let price = document.querySelector("#price"+id).value;
    console.log(price);
    App.contracts.FantasyLeague.deployed().then(function(instance){
      return instance.setPlayerPrice(id, web3.toWei(price, "ether"),
{gas: 3000000});
    }).then(function(result){
```

```
    console.log(result);
    //location.reload();
  }).catch(function(error) {
    console.warn(error);
  });
},

addAirDrop : function(){
  let account_address = jQuery("#send-token-to").val();
  let tokens_count = parseInt(jQuery("#send-token-count").val());
  App.fantasyLeagueInstance.sendToken(account_address,
web3.toWei(tokens_count, "ether")).then(function(){
    alert("Token Sent Successfully");
    location.reload();
  });
}

};


function renderPlayer(playerDetails){
let id =  playerDetails[0];
let name = playerDetails[1];
let price = web3.fromWei(playerDetails[2] , 'ether');
let is_sellable = playerDetails[3];
let ownedBy = playerDetails[4];

let is_own_player = ownedBy == App.account

let $parentDiv = is_own_player ? jQuery("#myPlayersList") :
jQuery("#otherPlayersList");

let $playerDiv = jQuery("<div class='player-details'></div>");
let $name = jQuery("<div><div>Name:</div><div> "+ name
+"</div></div>");
let $id = jQuery("<div><div>Id: </div><div>"+ id +"</div></div>");
let $price = jQuery("<div><div>Price(EPL token): </div><input
contenteditable=true id=price"+id+" onkeyup=\"App.updatePrice(" + id
+",event)\" value="+price+"></input></div>");

$playerDiv.append($name);
$playerDiv.append($id);
$playerDiv.append($price);
```

```
  if(is_own_player){
   $playerDiv.append(jQuery("<div>Sellable <label
class=\"switch\"><input id=sellableChkBox"+id+" type=\"checkbox\"
checked="+is_sellable+" onclick=\"App.changeSellable("+id+")\"><span
class=\"slider round\"></span></label> </div>"));
  }
 else{
   $playerDiv.append(handleBuy(is_sellable, price, id));
 }

 $parentDiv.append($playerDiv);
 //todo change
 if(document.querySelector("#sellableChkBox"+id)){
   document.querySelector("#sellableChkBox"+id).checked= is_sellable;
 }
 }

function handleBuy(is_sellable, price, id){
 if(is_sellable){

   return jQuery("<button class='button-17' role='button' playerID="+
id +" onclick='App.buyPlayer(" +id+", "+ price+")'\>Buy
Player</button>");
 }
 else{
   return jQuery("<div>Player Not Available for purchase</div>");
 }
}

function setTeamDetails(teamDetails){
 jQuery(".team-name").html(teamDetails[1]);
}

function setTokenBalance(balance){
 let token_balance = "<b>Token Balance(EPL): </b>" +
web3.fromWei(balance, "ether");
 jQuery(".team-token-balance").html(token_balance);
}

function setAccountBalance(){
 web3.eth.getBalance(App.account, function(err, result) {
   if (err) {
     console.log(err)
```

```
    } else {
      let balance = "<b>Balance(Eth): </b>" + web3.fromWei(result,
"ether");
      jQuery(".team-balance").html(balance);
    }
  });
}

$(function() {
  $(window).load(function() {
    App.init();
  });
});
```

References:

1. Blockchain in Action by Bina Ramamurthy
2. Web3 docs: https://web3js.readthedocs.io/en/v1.7.3/
3. Jquery: https://api.jquery.com/
4. UI elements: Toggle switch https://www.w3schools.com/howto/howto_css_switch.asp
5. Truffle doc: https://trufflesuite.com/docs/truffle/
6. Sequence/UML diagram: Lucidcharts
7. DApp tutorial/reference app: https://dapp
8. DApp reference: Election -Bina ramamurthy
9. Infuria for deployment to roptsen: https://infura.io/ & https://docs.infura.io/infura/getting-started
10. toString functionality in solidity: toString