Soccer Club Management and Fan Engagement Voting

1. Abstract

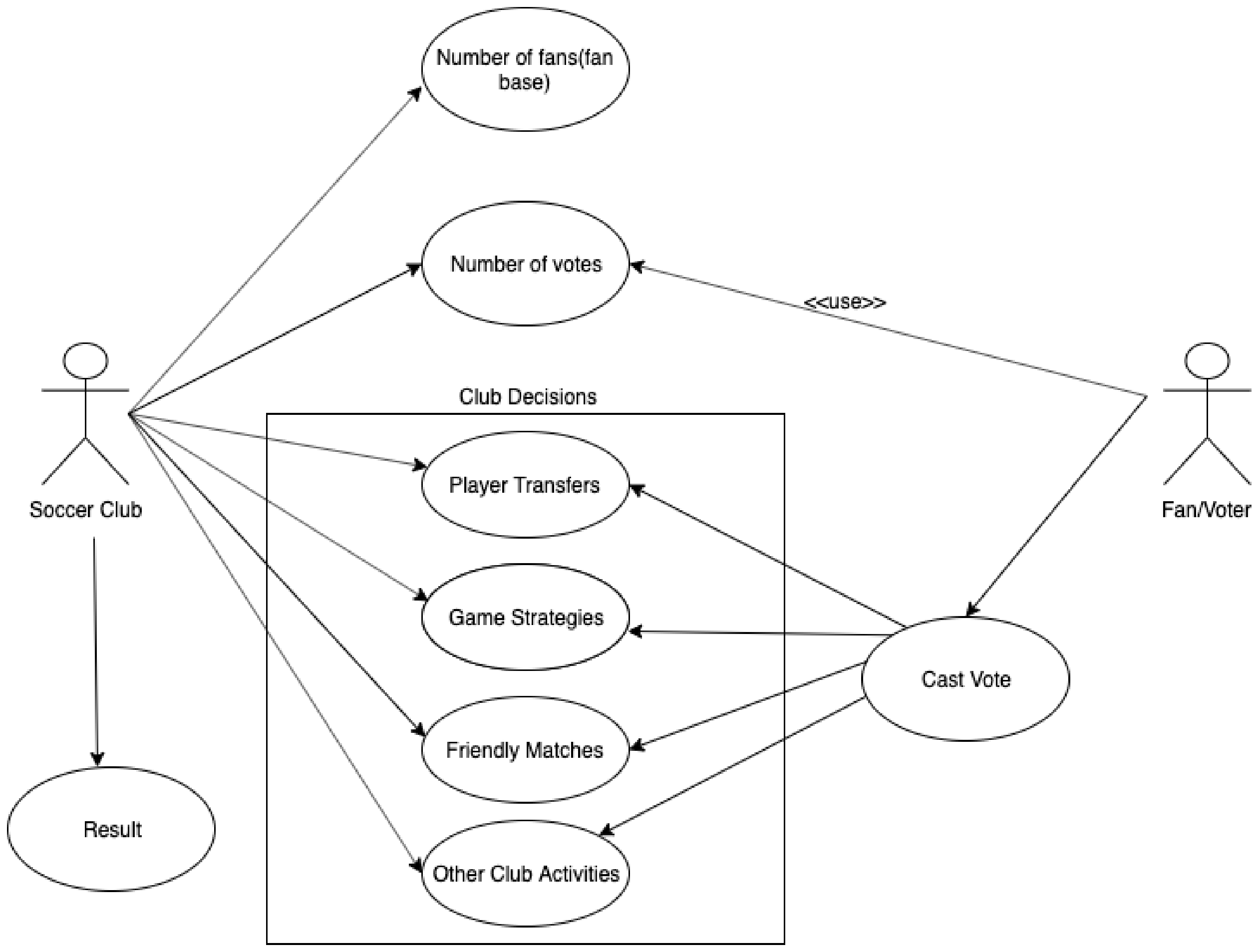
Title: Soccer club Management and Fan Engagement Voting

Details: Supporters are life blood of any sporting club and success attracts more fans. More supporters means more revenue and more revenue means better players and more success. Not only the soccer clubs but also the other sport clubs enjoy fan engagement towards their teams. Fan engagement depends upon their supporting team performance in the game. There will be many soccer club decisions that will decide the team's success in the game. Some of the main soccer club decisions such as player transfers, game tactics will have a deep impact on the team's performance. But what if a fan supporting team loses and having the tough time due to those decisions? There will be decrease in the fans support towards the losing club. The idea is to how a soccer club can increase the fan engagement for their respective team. Clubs can give fans a proposal to vote on various aspects of a club governance, such as who they play in friendlies and which players they should look to sign. In short, the club can ask the fans how the club is running. Blockchain can help in maintaining the voting integrity in this case and guarantees the transparency of the votes and creates a legal representation of the digital assets. In this approach clubs can select the fan base/number of fans and distribute the number of votes among the fans. Clubs will provide their decisions such as the strategies, player transfers, club governance and fans will vote on those decisions. The hope is that this will increase the fan engagement and they will not feel the physical proximity to their team to exert the influence of the decisions made by them. Also, the clubs will gain the advantage of increase in their team's performance.

Link: https://www.forbes.com/sites/stevemccaskill/2018/09/12/can-blockchain-give-fans-a-meaningful-say-in-how-soccer-clubs-are-run/

Name: Sai Kumar Aindla
Person Number[redacted]

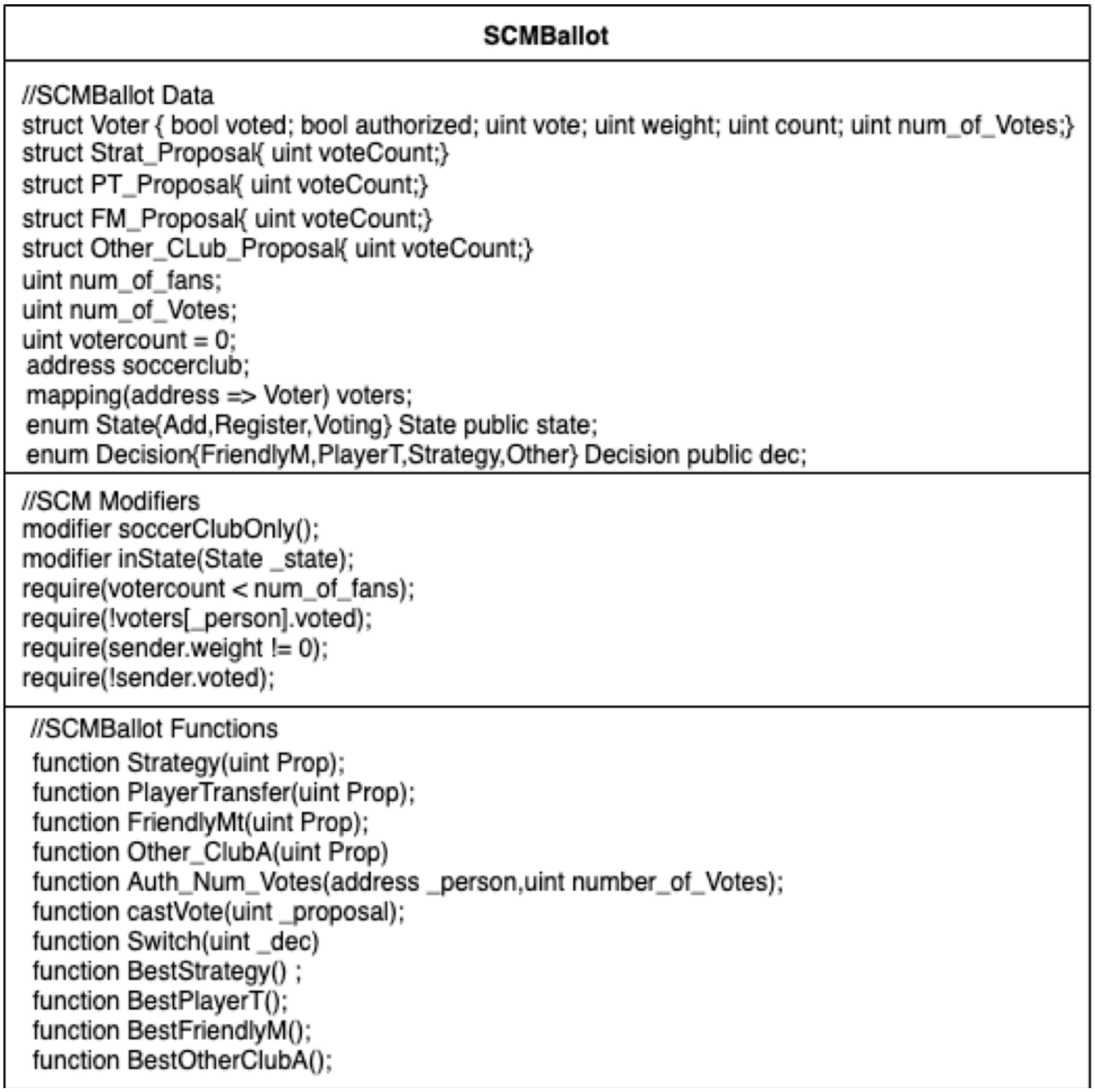
2. Use Case Diagram



In this use case diagram, the roles of actors are as follows:

1) Soccer Club (Owner)
   a) Authorizing number of fans who can vote.
   b) Assigning the votes to fans. The votes for each fan can be different and will be assigned by the Soccer club.
      Ex: one fan can contain 4 votes, and other may contain 2 votes.
   c) Giving the club decisions to fans for voting.
   d) Providing the number of proposals for each club decision.
   e) Select the Best proposal in each club decision.
2) Fan (Voter)
   a) A fan can utilize his votes and can distribute his votes to select any club decision proposal based on his preference.
      Ex: If a fan has been assigned 4 votes, he can use 2 votes to select a proposal in Strategy, 1 vote to select a proposal in Player Transfer and 1 vote to select any other club decision.

3. Contract Diagram



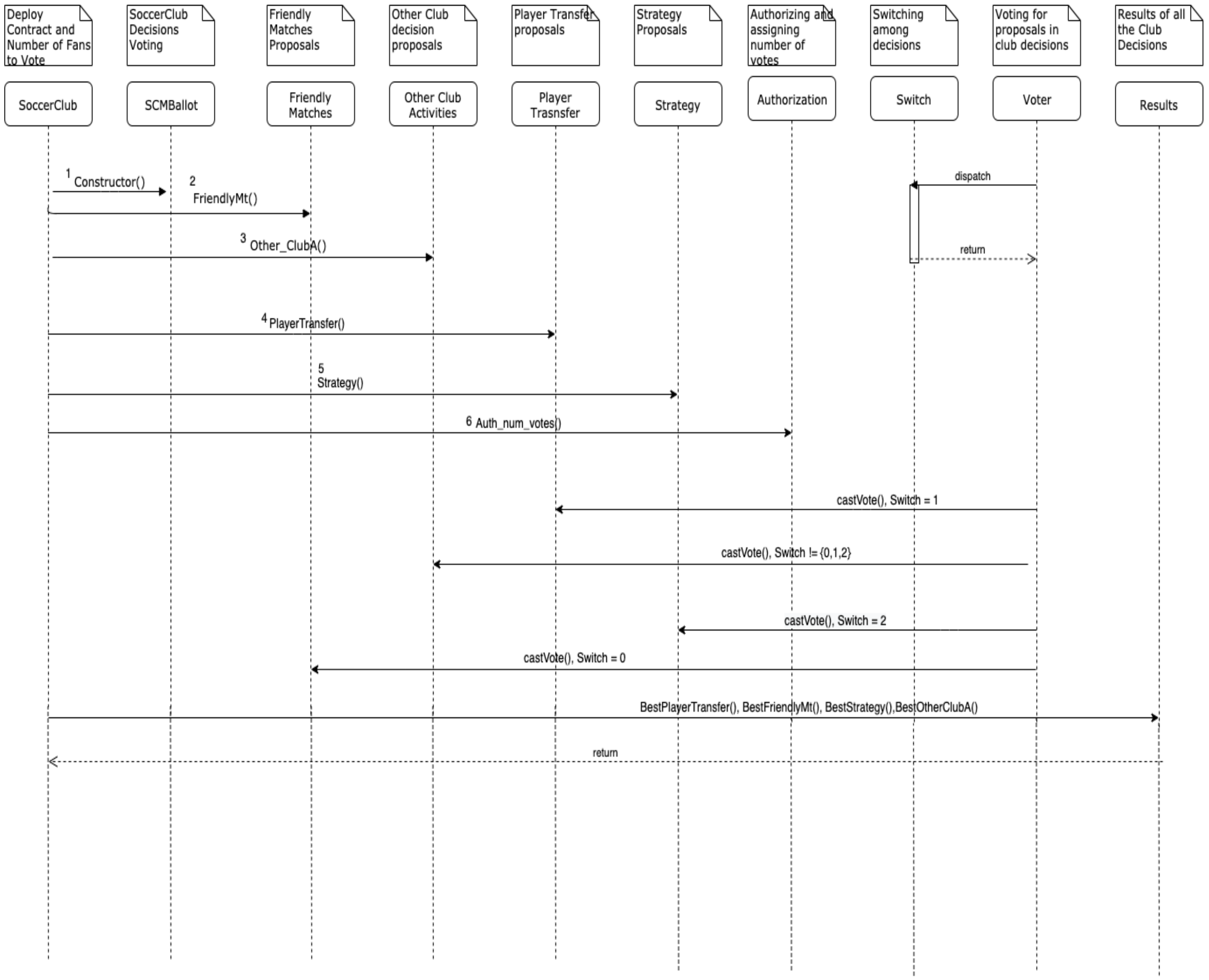
SCMBallot

```
//SCMBallot Data
struct Voter { bool voted; bool authorized; uint vote; uint weight; uint count; uint num_of_Votes;}
struct Strat_Proposal{ uint voteCount;}
struct PT_Proposal{ uint voteCount;}
struct FM_Proposal{ uint voteCount;}
struct Other_CLub_Proposal{ uint voteCount;}
uint num_of_fans;
uint num_of_Votes;
uint votercount = 0;
address soccerclub;
mapping(address => Voter) voters;
enum State{Add,Register,Voting} State public state;
enum Decision{FriendlyM,PlayerT,Strategy,Other} Decision public dec;
```

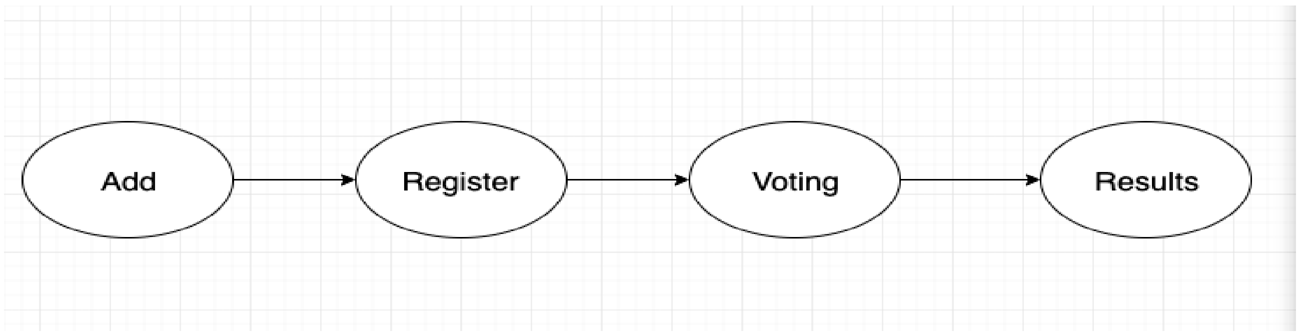
```
//SCM Modifiers
modifier soccerClubOnly();
modifier inState(State _state);
require(votercount < num_of_fans);
require(!voters[_person].voted);
require(sender.weight != 0);
require(!sender.voted);
```

```
//SCMBallot Functions
function Strategy(uint Prop);
function PlayerTransfer(uint Prop);
function FriendlyMt(uint Prop);
function Other_ClubA(uint Prop)
function Auth_Num_Votes(address _person,uint number_of_Votes);
function castVote(uint _proposal);
function Switch(uint _dec)
function BestStrategy() ;
function BestPlayerT();
function BestFriendlyM();
function BestOtherClubA();
```

4. Sequence Diagram

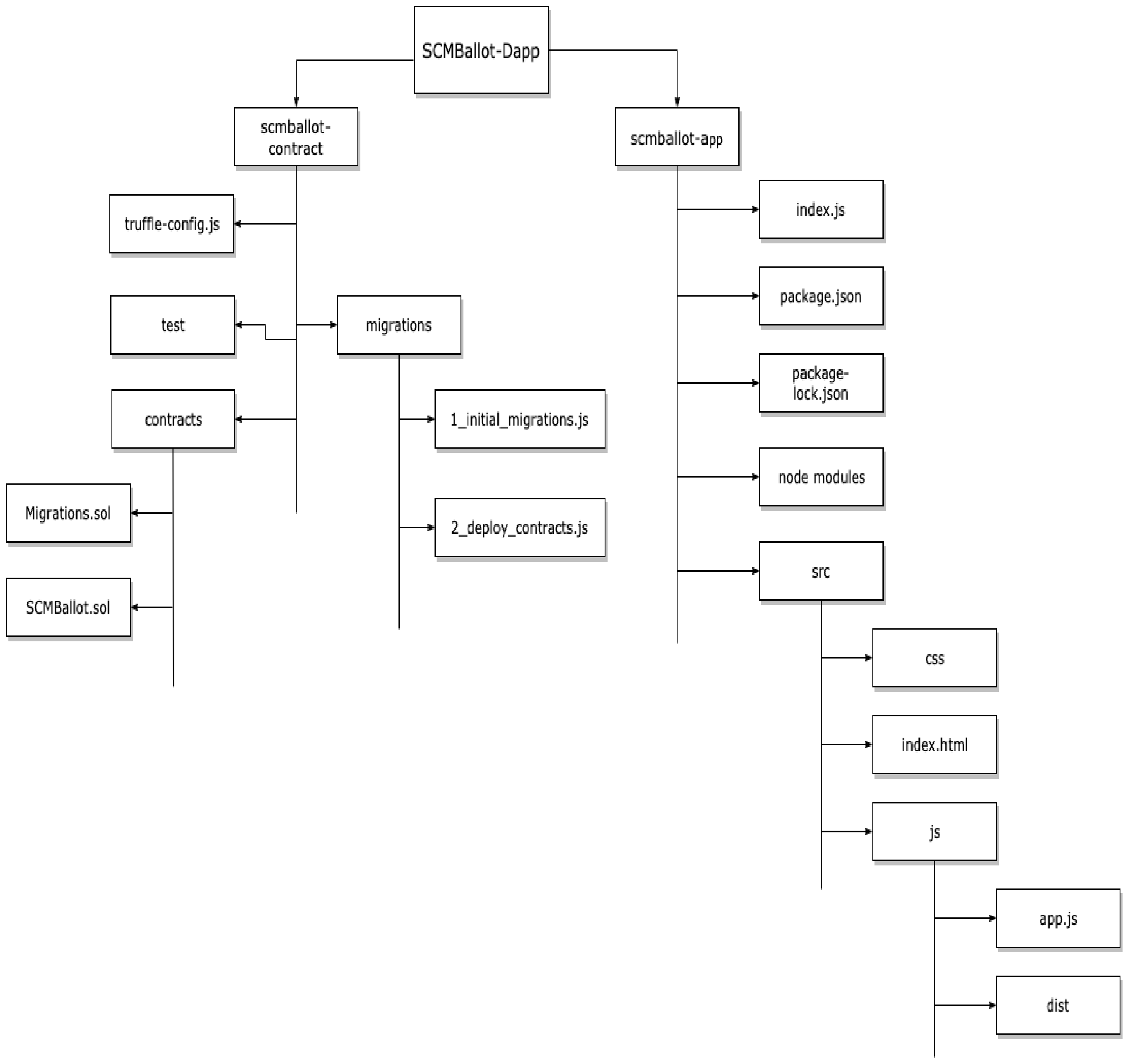

Deploy Contract and Number of Fans to Vote
SoccerClub Decisions Voting
Friendly Matches Proposals
Other Club decision proposals
Player Transfer proposals
Strategy Proposals
Authorizing and assigning number of votes
Switching among decisions
Voting for proposals in club decisions
Results of all the Club Decisions
SoccerClub
SCMBallot
Friendly Matches
Other Club Activities
Player Trasnsfer
Strategy
Authorization
Switch
Voter
Results
1 Constructor()
2 FriendlyMt()
3 Other_ClubA()
4 PlayerTransfer()
5 Strategy()
6 Auth_num_votes()
dispatch
return
castVote(), Switch = 1
castVote(), Switch != {0,1,2}
castVote(), Switch = 2
castVote(), Switch = 0
BestPlayerTransfer(), BestFriendlyMt(), BestStrategy(),BestOtherClubA()
return

5. State Diagram



1) States start from Adding the proposals, Registering the voters, Fans Voting and Results declaration by owner.
2) Along with these, 4 club decisions are also taken as states so that can fans freely switch to other decisions and vote them. The switching completely depends upon the interest of the fans so the state diagram for these states is not possible.

6. Architecture Diagram



The above architecture diagram is what I have developed during the entire lab1. For the first part of lab1 I have developed scmballot-contract solidity code. In the second part, a decentratlized web application is built using the smart contract scmballot code.

scmballot-contract

1) Directory "scmballot-contract" consists of the configurations and smart contract code for deploying it on the truffle ganache.
2) "SCMBallot.sol" is the smart code for the complete project.
3) Truffle configurations such as port number, network id are present "truffle-config.js".
4) For deploying the smart contract, we need to create a "2_deploy_contracts.js" and have to give any deployment value (in my case "Number of Fans" who can vote given by the owner while deploying the code).

scmballot-app

1) The Dapp is developed using vanilla Javascript and Express server. Smart contract is deployed on the Ganache test network using the Metamask injected web3.
2) The starting point of the application is "index.js", where the details of server is mentioned.
3) Landing page for web app is "index.html".
4) Server dependencies are included in "package.json".
5) "App.js" is the main code where the interaction of webapp and smart contract takes place.
6) Directory "dist" contains "web3", "truffle-contract" and "bootstrap" javascript files.
7) Styling to web page is added using the "css" directory, which consists of style and bootstrap files.

7. Workflow Instructions or ReadMe

Prerequisites

Before using the web application, the following packages and tools are required for the development

a. Ganache
b. npm
c. Metamask
d. Truffle

Steps for working on web application

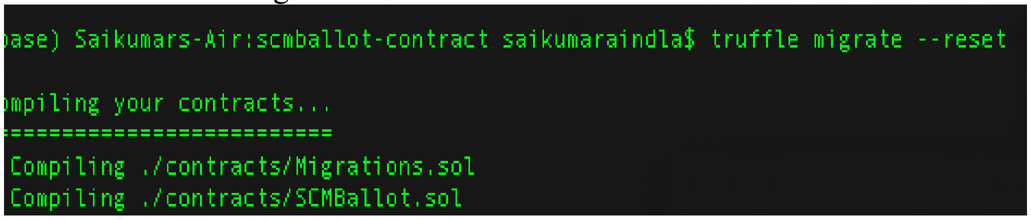
Please follow the below instructions for the workflow of the web application.

1. Unzip the file SCMBallot- Dapp, there will be two folders in the directory SCMBallot-Dapp.

2. First step is to open the ganache in your system, before deploying the smart contract code.

3. Next, open the chrome browser and login in to Metamask by importing the seed of Ganache.

4. Now, open the terminal/cmd and navigate to the "scmballot-contract" directory, run the command "truffle migrate --reset".



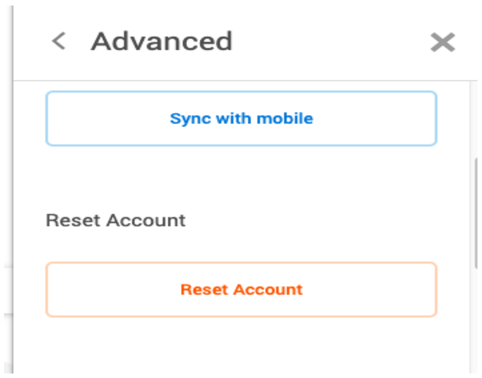
```
base) Saikumars-Air:scmballot-contract saikumaraindla$ truffle migrate --reset

mpiling your contracts...
=========================
 Compiling ./contracts/Migrations.sol
 Compiling ./contracts/SCMBallot.sol
```

5. After the running the command, you can see a "build" folder created in the "scmballot-contract" directory. This folder consists of the json file of Smart contract.

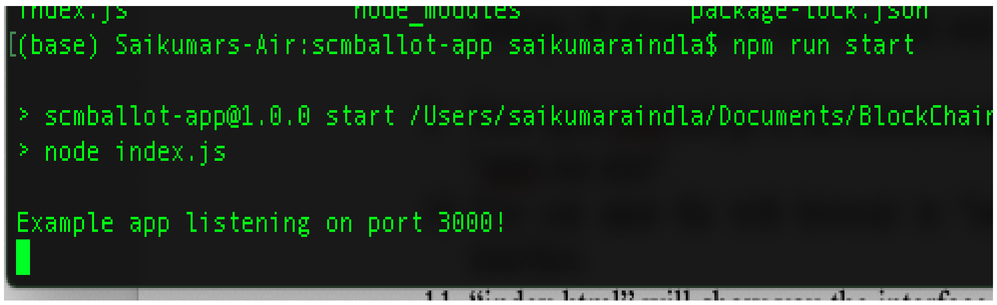
6. In next step, go to the "scmballot-app" in cmd/terminal and run the command "npm install". This will install all the dependencies required for the project.

7. In next step, open the browser and reset the accounts in the metamask wallet.

8. Every time, you get an payload error or gas limit error, reset the account by going to settings → advanced→ reset account and try again.



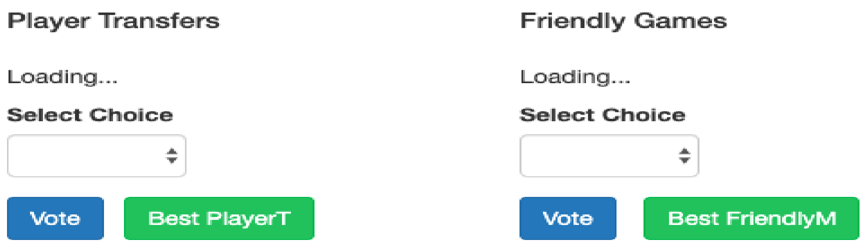
9. Now open cmd and go to the frontend part, i.e "scmballot-app" and run the command "npm run start".



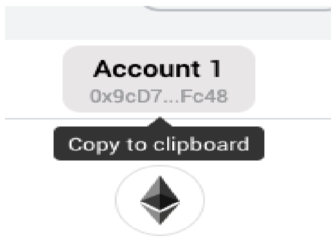
```
 index.js                node_modules            package-lock.json
[(base) Saikumars-Air:scmballot-app saikumaraindla$ npm run start

 > scmballot-app@1.0.0 start /Users/saikumaraindla/Documents/BlockChai
 > node index.js

Example app listening on port 3000!
```

10. You can open the web browser in "localhost:3000" and see the web application interface.

11. "index.html" will show you the interface of the application.
12. When you open the browser showing the proposals of the 4 club decisions as "Loading". They will be filled, once you give the numbers of proposals for each decision and click on submit.



13. Before entering the number of proposals for each decision, check the address in the Metamask which you deployed the contract with, that will be owner account or main address where the metamask is deployed.

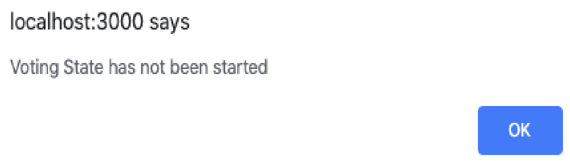


14. Check the state of the project by clicking the button "Actual State", an alert will be popped up saying "**You are in Add state"**, which is the first step in the project.

localhost:3000 says

You are in Add State

15. Also, check the state of decision by clicking the button "Decision State", that says "**Voting State has not been started**". This state will only come into role during the "Voting" phase is initiated.

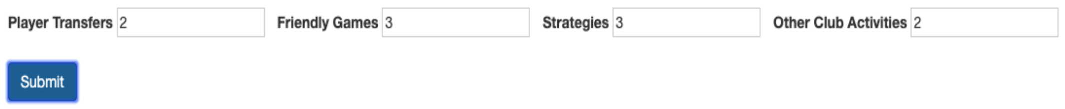


localhost:3000 says

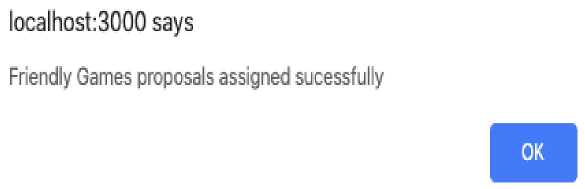
Voting State has not been started

16. All the states are automatically instantiated when one phase comes to end depending upon the modifiers in the smart contract.

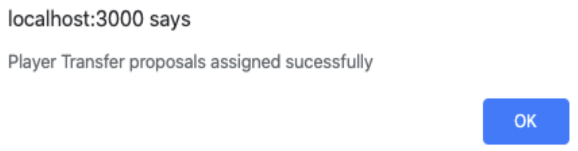
17. Now enter the number of proposals for each decision as below and click the "Submit" button.
    Player Transfer – 2
    Friendly Games -3
    Strategies – 3
    Other Club Activities – 2


Player Transfers 2 Friendly Games 3 Strategies 3 Other Club Activities 2

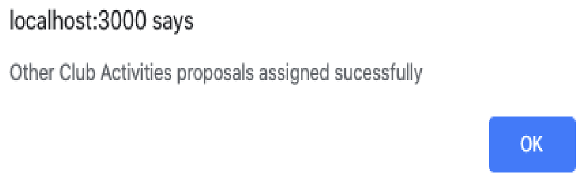
18. After clicking the button, metamask will ask for the confirmation of the 4 transactions, confirm one by one, you can see that the below section is be populated with the number of proposals you gave for each decision. Sample is shown below.


localhost:3000 says

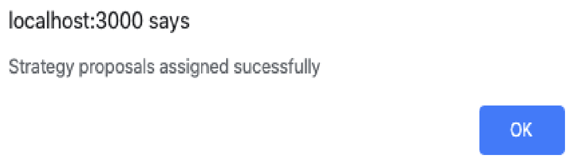
Friendly Games proposals assigned sucessfully


localhost:3000 says

Player Transfer proposals assigned sucessfully

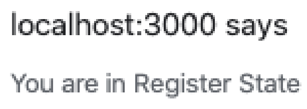

localhost:3000 says

Other Club Activities proposals assigned sucessfully


localhost:3000 says

Strategy proposals assigned sucessfully

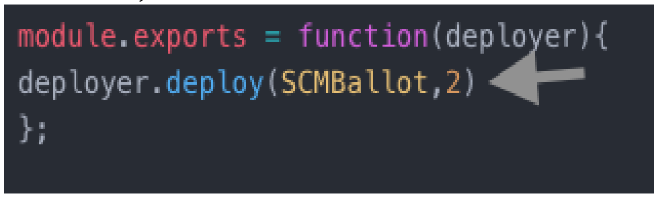
19. Now click on "Actual State" button, you will see that "**You are in Register State"**, that means once confirm all the proposals for each decision, the state has changed to register.


localhost:3000 says

You are in Register State

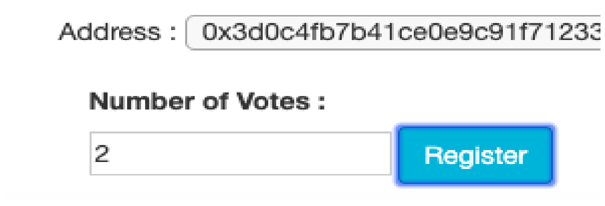
20. Now the owner can register the number of accounts (fans) and assign the number of votes to each account, who will vote.

21. Here, the number of fans are decided during the deployment of smart contract in "**2_deploy_contracts.js**". I have given the number of fans as 2, so only we can register two voters, who can vote.



```
module.exports = function(deployer){
deployer.deploy(SCMBallot,2)
};
```

22. Now coming to the register phase, you can see below there will be a drop down list of addresses. Select the first address Starting with "0x3dc…" and assign him the number of votes as '2' and click the register button. Metamask will ask you to confirm the transaction for registration process.

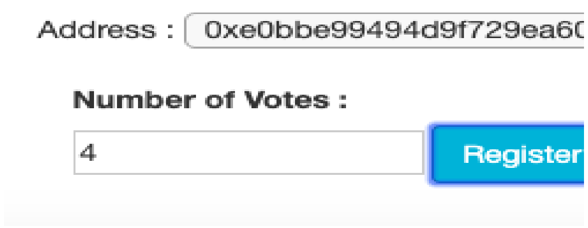


Address : 0x3d0c4fb7b41ce0e9c91f71233

Number of Votes :

2

23. Now select the second address starting with "0xe0bb…" and assign him the number of votes as '4' and click the register button. Metamask will ask you to confirm the transaction for registration process.

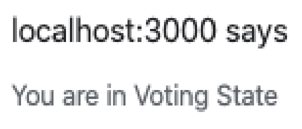


Address : 0xe0bbe99494d9f729ea6

Number of Votes :

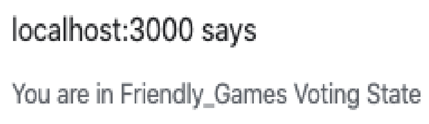
4

24. Now click on the "Actual State" button, it says "**You are in Voting State"** and also click on the button " Decision State", Now you can see that it says you are in "**You are in Friendly_Games Voting State"**.



localhost:3000 says

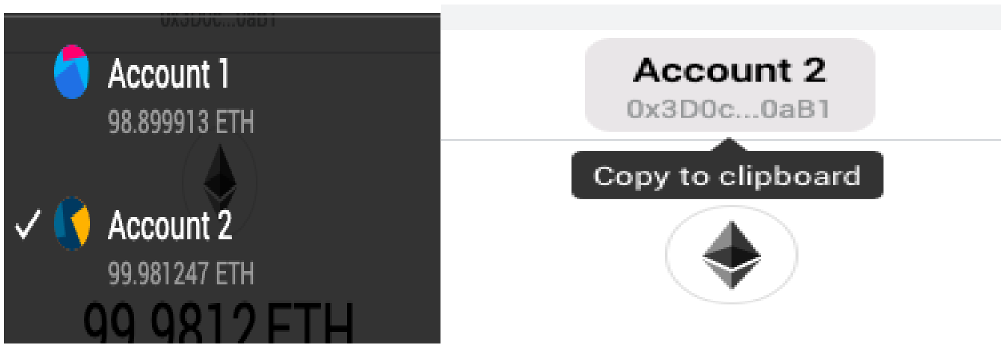
You are in Voting State



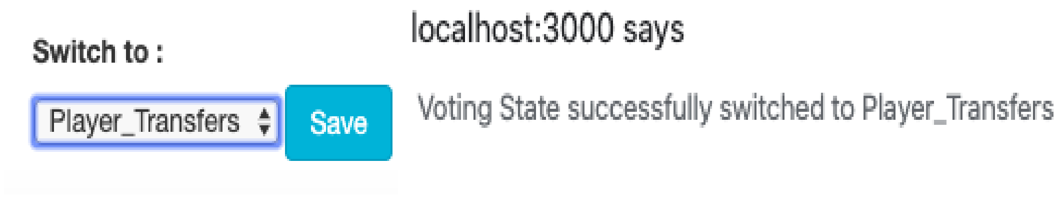
localhost:3000 says

You are in Friendly_Games Voting State

25. Now start the voting process, since only two voters are present. Open the Metamask wallet and go to the account with address of first voter ""0x3dc…" as shown below.



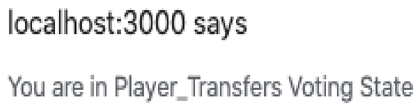
26. Start the voting process by switching the voting state to "Player Transfers" by using the drop-down list of "Switch to" and click on the save button. Confirm the metamask transaction. You will get an alert saying the voting state has been switched.



localhost:3000 says

Voting State successfully switched to Player_Transfers

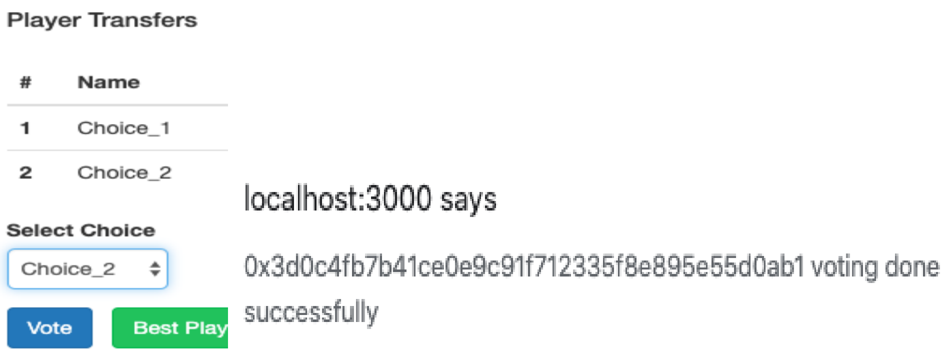
27. Now verify, by clicking the "Decision State", it shows "**You are in Player_Transfers Voting State".**



localhost:3000 says

You are in Player_Transfers Voting State

28. Select the "Choice 2" Player Transfers and click the vote button below it. Confirm the metamask transaction and alert will raised saying "**Voting done successfully**".



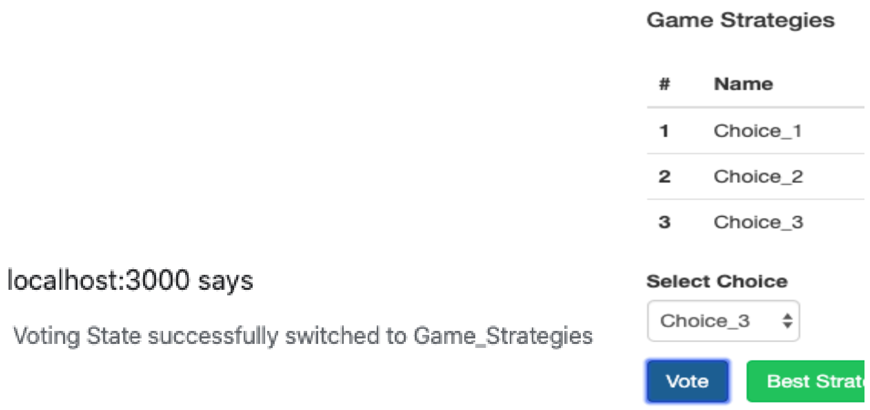
Player Transfers

#	Name
1	Choice_1
2	Choice_2

localhost:3000 says

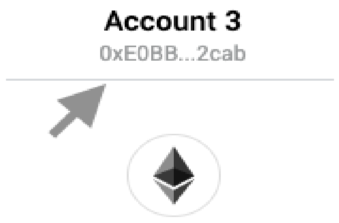
0x3d0c4fb7b41ce0e9c91f712335f8e895e55d0ab1 voting done successfully

29. Now switch state as showed above to "Game Strategies" by clicking the "Save" button and vote for the "Choice 3". Confirm the transaction and alert will be raised saying "**Voting done successfully**".

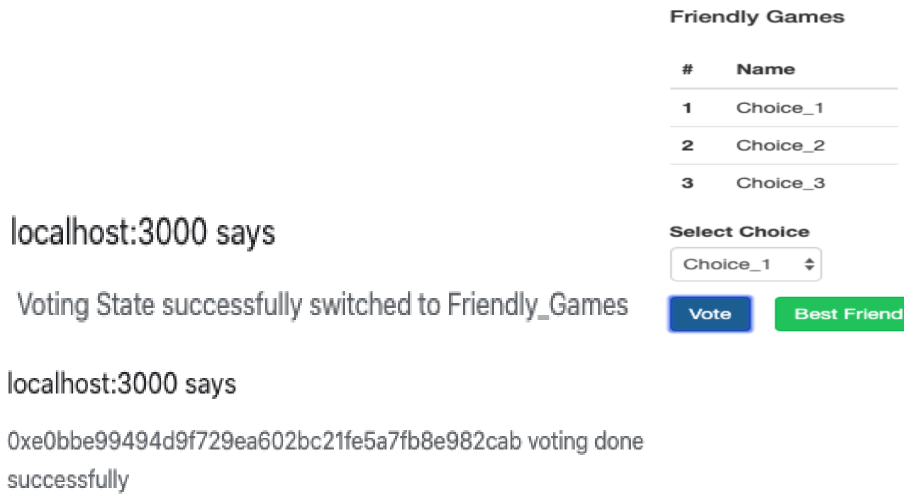

localhost:3000 says

Voting State successfully switched to Game_Strategies

30. The voting of first user is done.
31. Select the second address in metamask wallet "0xe0bb…" and start in the same procedure as above.



32. After each vote confirm the transaction and you can check current phase and decision state by clicking the buttons "Actual State" and "Decision State".
33. Switch to "Friendly Games" and vote for the "Choice 1" two times by clicking the vote button twice.

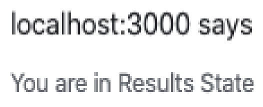

localhost:3000 says

Voting State successfully switched to Friendly_Games

localhost:3000 says

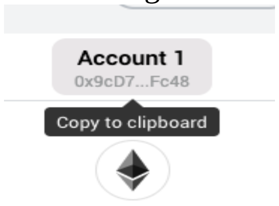
0xe0bbe99494d9f729ea602bc21fe5a7fb8e982cab voting done successfully

34. Similarly, Next switch to "Other Club Actitvities" and Click on "Save" button and vote for the "Choice 2" two times. The outputs will be similar to the above process.
35. Click on "Actual State" button, it says "**You are in Results State".**

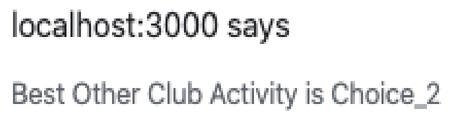

localhost:3000 says

You are in Results State

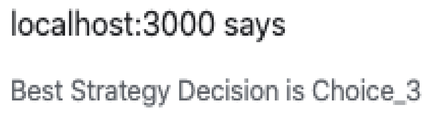
36. Results can only be declared by the account who deployed or owner so, open metamask wallet and go to the main account "0x9c..".



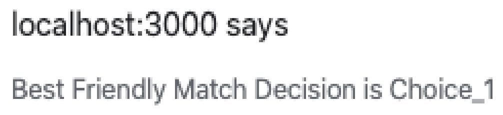
37. Next Click the buttons, "Best Player T", "Best FriendlyM", "Best Strategy", "Best OtheClubA". After each click, confirm the transaction.
38. Alerts will be raised which will be give the results for each decision.


localhost:3000 says

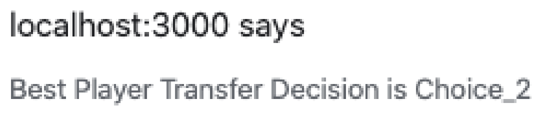
Best Other Club Activity is Choice_2


localhost:3000 says

Best Strategy Decision is Choice_3


localhost:3000 says

Best Friendly Match Decision is Choice_1


localhost:3000 says

Best Player Transfer Decision is Choice_2