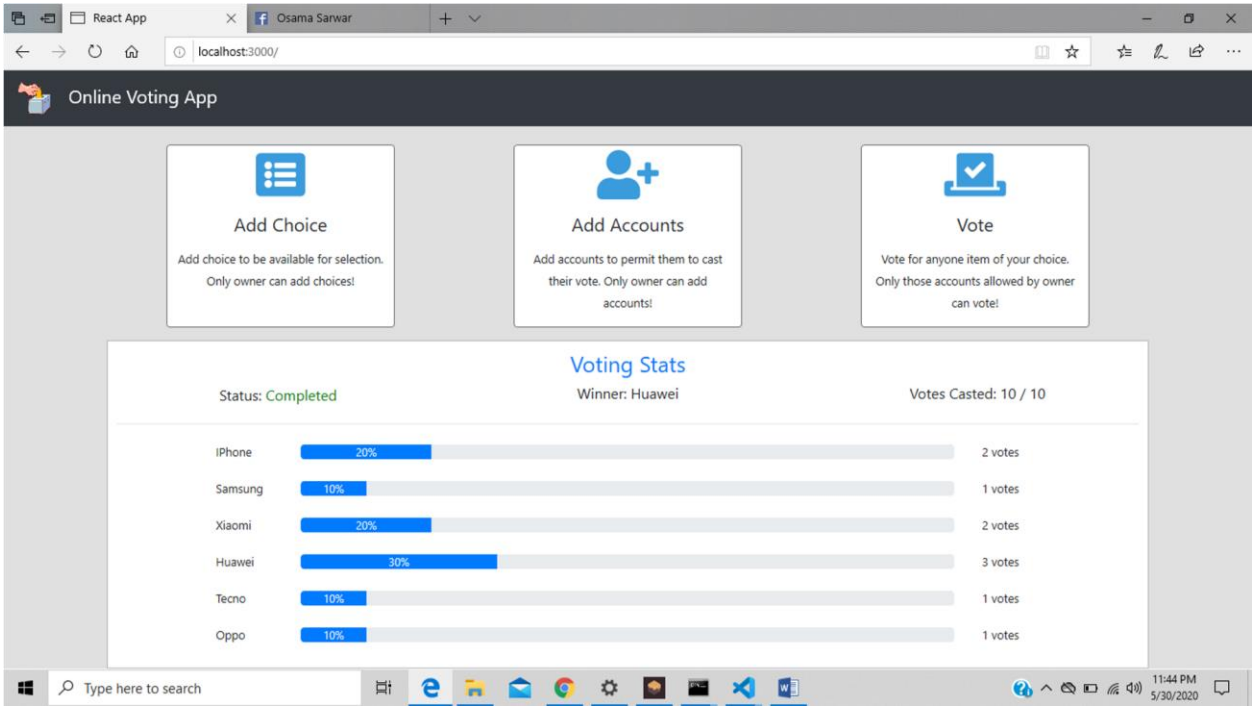
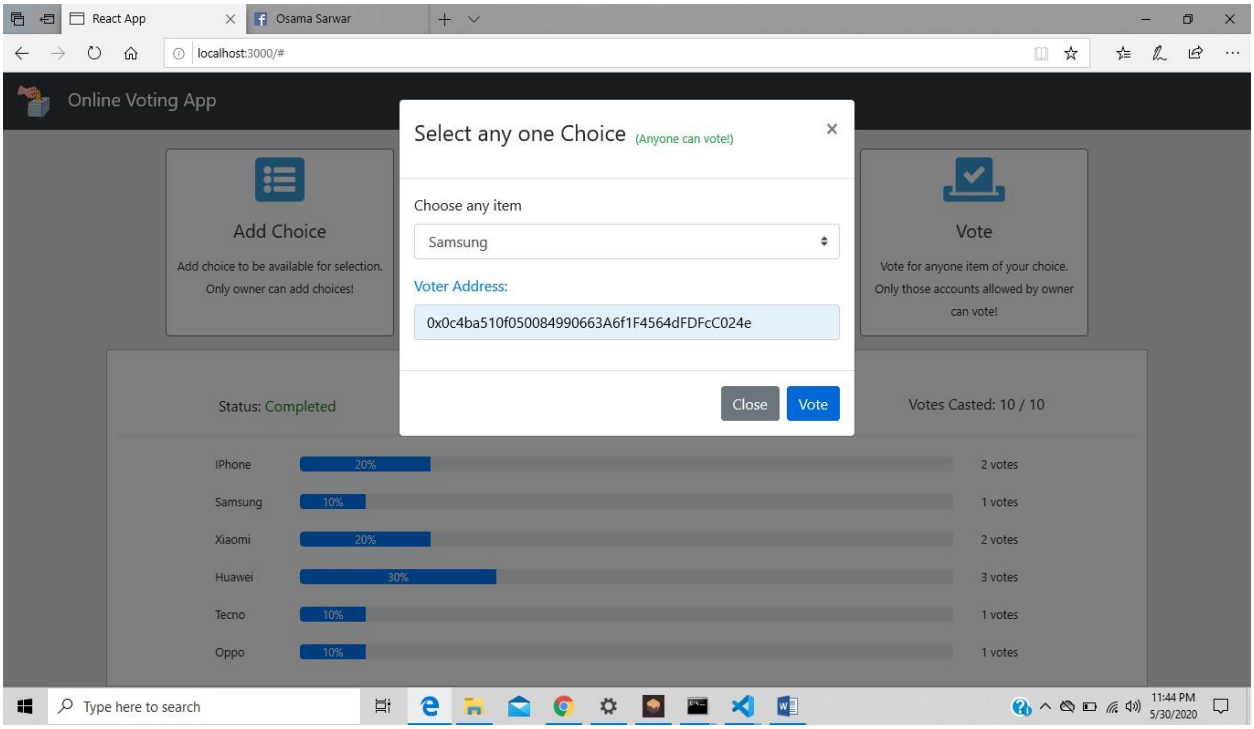
Usama Sarwar

Roll no: i160188

Screenshot of Voting Web DApp:





Requirements:

Req-1: There should be an option to add choices. Only owner could add choices.

Solidity:

```
    modifier onlyOwner {
        require(msg.sender == owner, "Only owner is allowed");
_;     }

    function addChoices (string memory c) public onlyOwner {
choices.push(c);          n_choices += 1;
voteChoices[c] = 0;



    }
```

To handle that requirement, I've created a function (addChoices) and added a modifier (onlyOwner) to it to verify that only owner is able to add choices. When the function is called with a string argument, that string is added to the choices array and number of votes against that choice is set to 0.

ReactJS: (Screenshots)

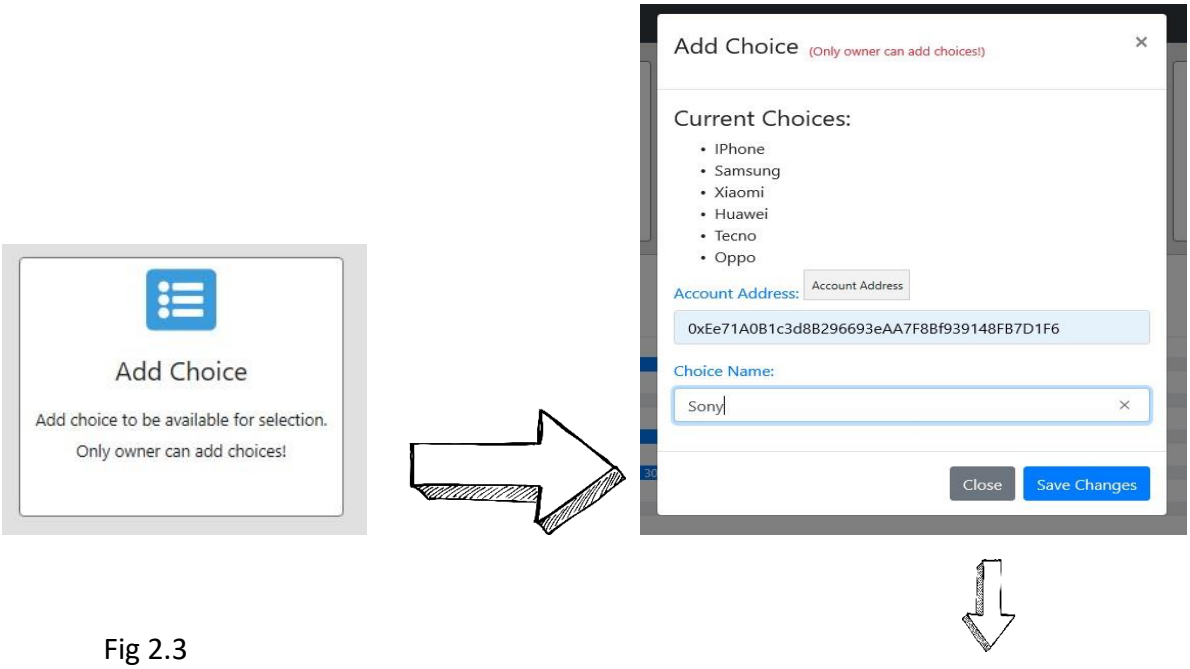


Fig 2.3

ReactJS: (Code)

```
    renderChoiceModal(){
return(

          <Modal show={this.state.choiceModalShow} onHide={this.handleClose}>
          <Modal.Header closeButton>
            <Modal.Title>Add Choice</Modal.Title>
            <p className="text-
danger" style={{marginTop:15, marginLeft:10, fontSize:13}}>(Only owner can add choices!)</p>
          </Modal.Header>
          <Modal.Body>
            <h4>Current Choices:</h4>
            <div>{this.renderChoices()}</div>
            <form name="accForm" onSubmit={this.addChoice}>
              <div className="form-group" role="form">
              <label className="text-primary">Account Address: </label>
                <input type="text" className="form-
control" name="accountAddInp" title="Account Address"></input>
            </div>
            <div className="form-group" role="form">
```

```
              <label className="text-primary">Choice Name: </label>
                <input type="text" className="form-control" name="choiceInp" title="Choice Name"></input>
              </div>
        </form>
          </Modal.Body>
          <Modal.Footer>
            <Button variant="secondary" onClick={this.handleClose}>
              Close
            </Button>
            <Button variant="primary" onClick={this.addChoice}>
              Save Changes
            </Button>
          </Modal.Footer>
        </Modal>
      );
     }

     renderChoices(){
return (
       <ul>
       {this.state.choices.map(choice => <li key={choice}>{choice}</li>)}
       </ul>
      );
   }



     addChoice = (event) => {
event.preventDefault();
      let acc=document.accForm.accountAddInp.value
let ch=document.accForm.choiceInp.value       let
err=false
      //alert(document.accForm.accountAddInp.value);
if (ch){
        this.state.simpcontract.methods.addChoices(ch).send({from: acc}).then((result) => {
          // console.log("Success! Got result: " + result);
const choiceTmp=this.state.choices.slice()
choiceTmp.push(ch)
          this.setState({choices:choiceTmp})

        }).catch((err) => {
```

```
          console.log("Failed with error: " + err);
alert(err)


        });


      }
     };
```

When the Add Choice Dialog is clicked the modal pops up on the page and asks for Owner address and choice input. Any invalid owner address entry displays Alert box with the Error Message. On successfully adding the choice, an entry of progress bar with the vote counts is added as shown in fig 2.3

Req-2: There should be an option to add accounts. Only owner could add accounts.

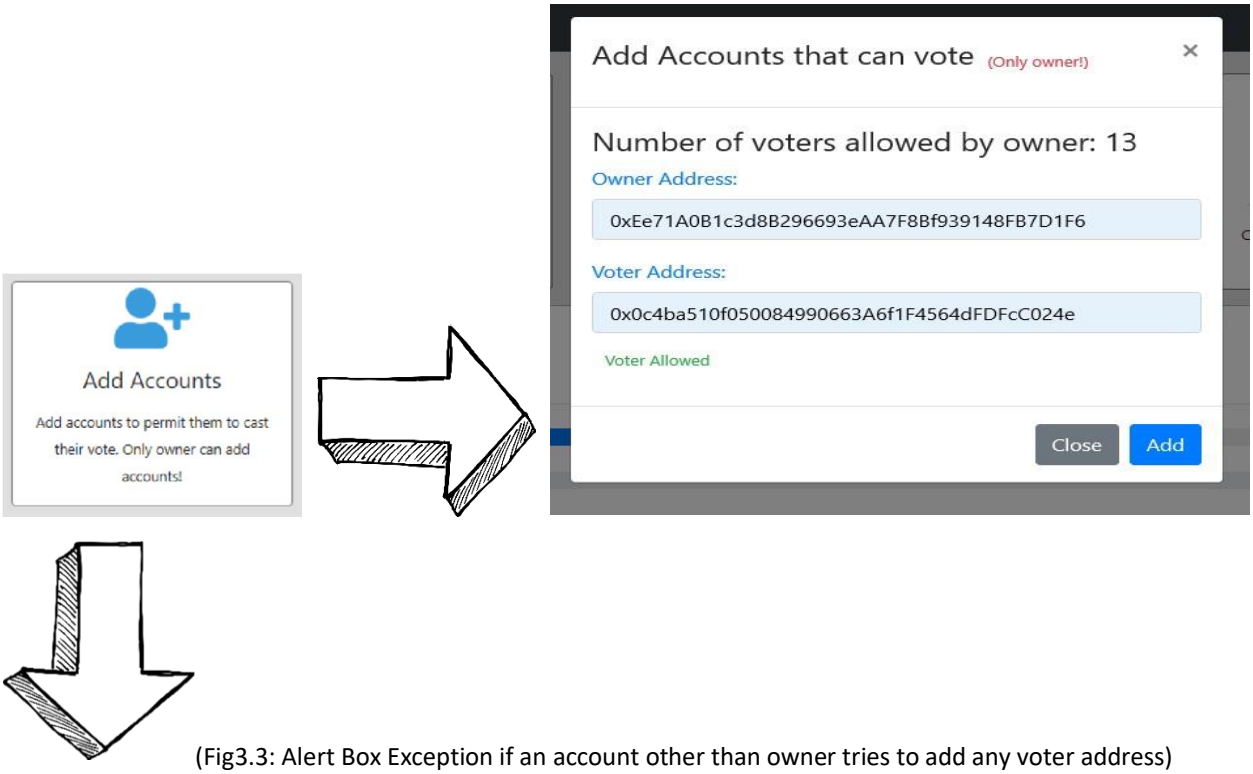
Solidity:

```
    mapping(address => string) public allowedAddresses;
```

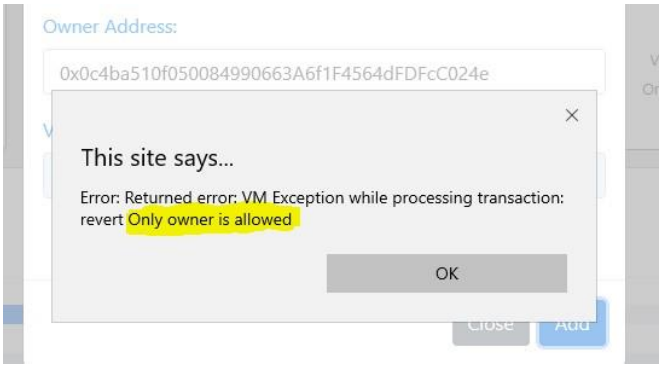
```
function addAddress(address adrs)public onlyOwner{
        if (!(keccak256(abi.encodePacked(allowedAddresses[adrs])) == keccak256(abi.encodePacked("Allowed")
))){
            n_voters += 1;
            allowedAddresses[adrs] = "Allowed";
        }
    }
```

addAddress function takes the address as parameter (which is a key in allowedAddresses mapping) and set the value of that address to “Allowed”. Only those addresses in the mapping whose value is set to “Allowed” are allowed to vote.

ReactJS: (Screenshots)



(Fig3.3: Alert Box Exception if an account other than owner tries to add any voter address)



ReactJS: (Code)

```
      addVoter = (event) => {
event.preventDefault();

      let ownerAddress=document.addVoterForm.ownerAddressInp.value
let voterAddress=document.addVoterForm.voterAddressInp.value       let
err=false

      //alert(document.accForm.accountAddInp.value);
if (voterAddress){

        this.state.simpcontract.methods.addAddress(voterAddress).send({from: ownerAddress}).then((result)
=> {

          // console.log("Success! Got result: " + result);
this.updateNVoters();

          //this.setState({n_voters: n_vtrs});


        }).catch((err) => {
          console.log("Failed with error: " + err);
alert(err)



        });


      //  const n_vtrs= this.state.simpcontract.methods.n_voters().call();
       // this.setState({n_voters:n_vtrs});


      }
     }

     async updateNVoters (){
      const n_vtrs =await this.state.simpcontract.methods.n_voters().call({from: this.state.account}, func
tion(error, result){});

      this.setState({n_voters: n_vtrs,vtrAllow:true});
     }
```

(Note: The renderAddAccountModal() function is pretty much same as the renderChoiceModal() function shown in Req1)

When the modal input addresses are entered and Add Button is clicked addVoter function is called which adds the address of voter if owner address is valid otherwise throws exception in the alert box as shown in fig3.3. This function further calls updateNVoters method to update number of allowed voters(n_voters) State variable.

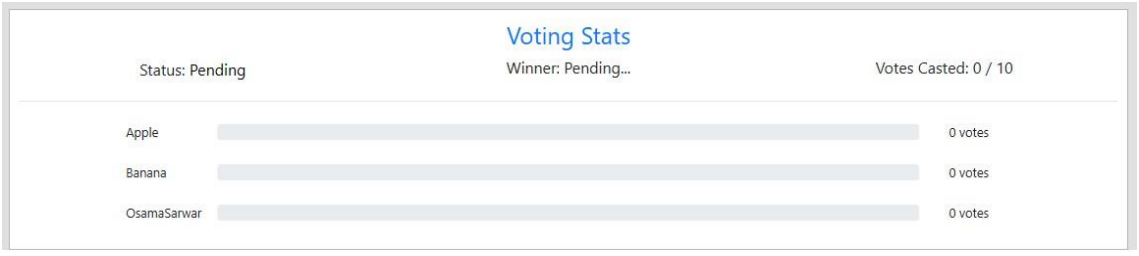
Req-3: Vote cannot be casted unless choices count reaches some threshold value.

Solidity:

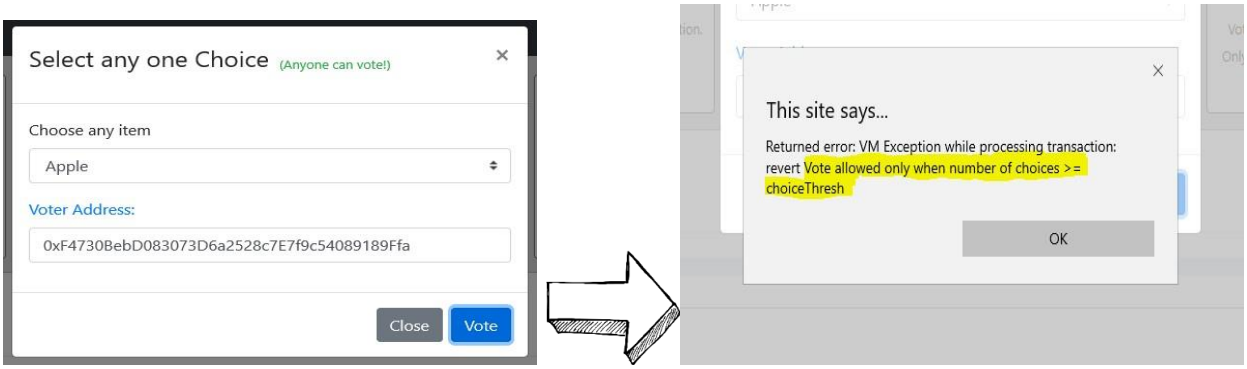
```
    modifier choiceThreshReached {
        require(n_choices >= choiceThresh, "Vote allowed only when number of choices >= choiceThresh");
_;     }

    function vote(uint256 choice_no) public votingNotCompleted choiceThreshReached addressCanVote {
votes[msg.sender] = choices[choice_no];         voteChoices[choices[choice_no]] += 1;

        uint256 currVotes = voteChoices[choices[choice_no]];
allowedAddresses[msg.sender] = "Voted";         if
(currVotes>maxVotes){             maxVotes = currVotes;
winningChoice = choice_no;

        }
        voteCount += 1;
    }
```

Whenever any voter will try to vote, the vote function won't be called if current number of choices added are less than required threshold. The modifier choiceThreshReacher determines if number of choices are greater than or equal to threshold.

ReactJS: (Screenshot)



Current number of choices: 3



As seen in the screenshots, voting is not allowed unless number of choices don't reach the threshold. The threshold in this case is set to 5.

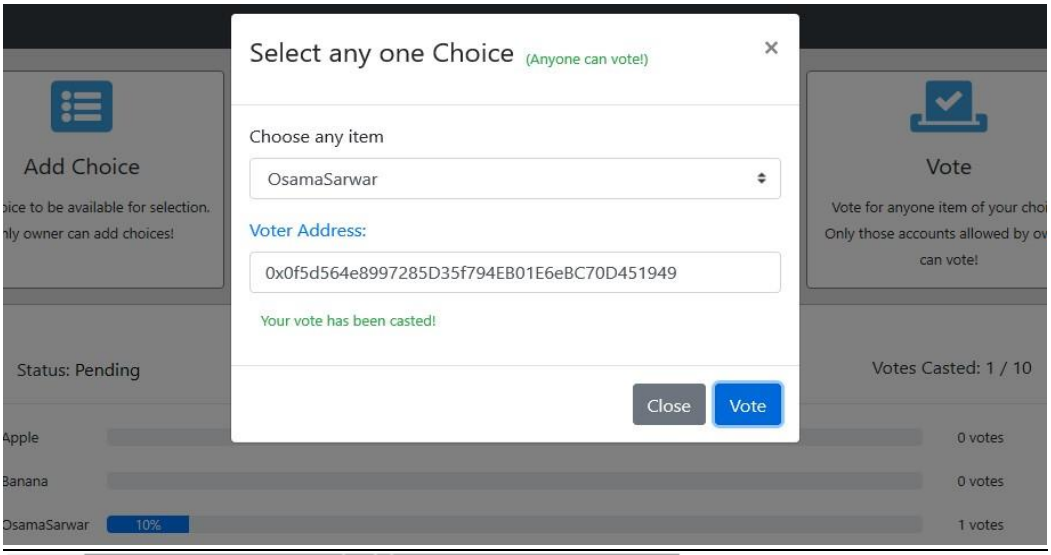
Req-4: An account can only vote ONCE.

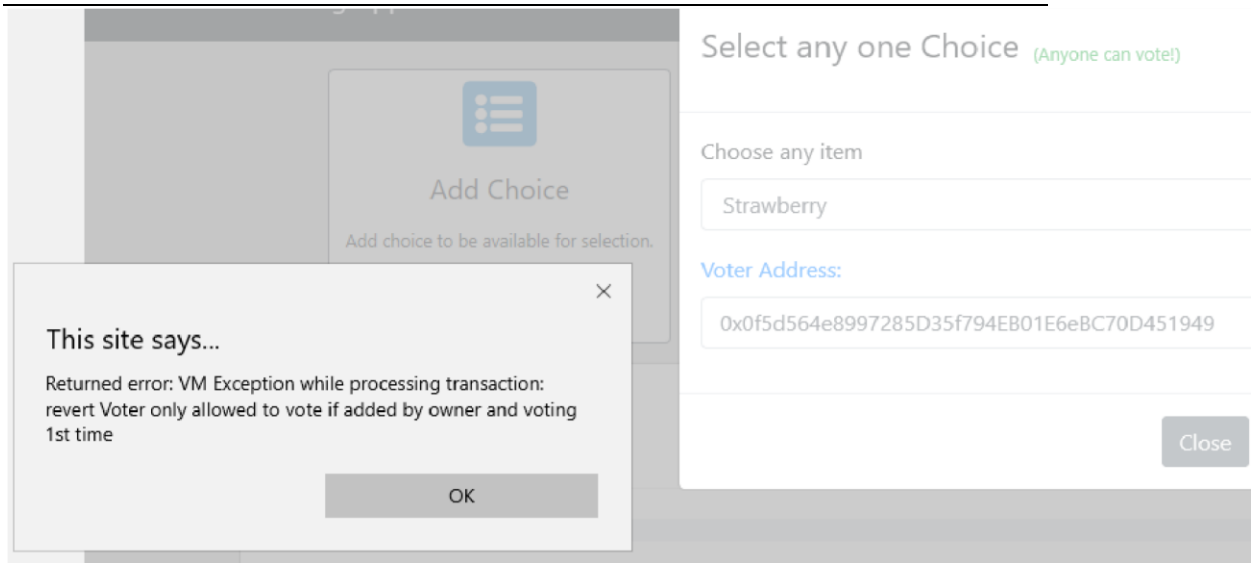
Solidity:

```
    function vote(uint256 choice_no) public votingNotCompleted choiceThreshReached addressCanVote {
votes[msg.sender] = choices[choice_no];         voteChoices[choices[choice_no]] += 1;
        uint256 currVotes = voteChoices[choices[choice_no]];

        allowedAddresses[msg.sender] = "Voted"; // ADDRESS VALUE CHANGED TO VOTED

        if (currVotes>maxVotes){
maxVotes = currVotes;
winningChoice = choice_no;
        }
        voteCount += 1;
    }
```

When the allowed user cast his vote, his value in the allowedAddresses mapping is set to "Voted" and only those addresses whose value is set to "Allowed" can cast their vote (specified in modifier addressCanVote). Therefore that particular address would not be able to vote again.

ReactJS: (Screenshot)





As seen when same voter tried to vote 2nd time, an alert box appears with message that "Voter only allowed to vote if allowed by owner and **VOTING 1ST TIME**"

Req-5: Once a total number of votes (choose some number) have been casted, the winner is chosen and anyone can then query for winner.

Solidity:

```
    modifier votingCompleted {
        require(voteCount>=voteThresh, "Voting Not Completed");
_;
    }
    function checkWinner () public view votingCompleted returns(string memory){
return choices[winningChoice];


    }
```

WinningChoice variable updates in vote() function each time vote count of a choice becomes maximum. checkWinner() function simply returns the winning choice when the voting completes(i.e votingCount becomes equal to votingThreshold that was already set).

ReactJS: (Screenshots)

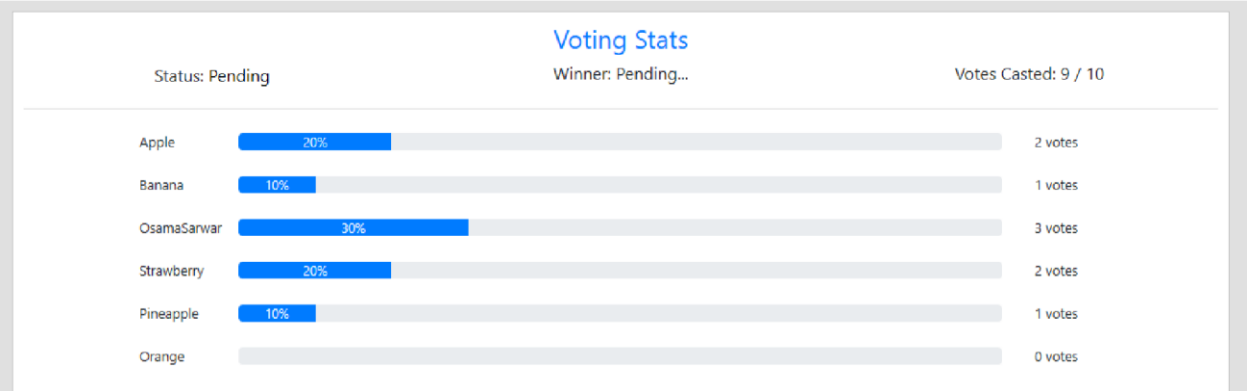
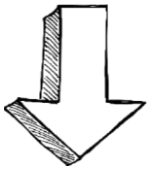


Fig5.1 Before final vote



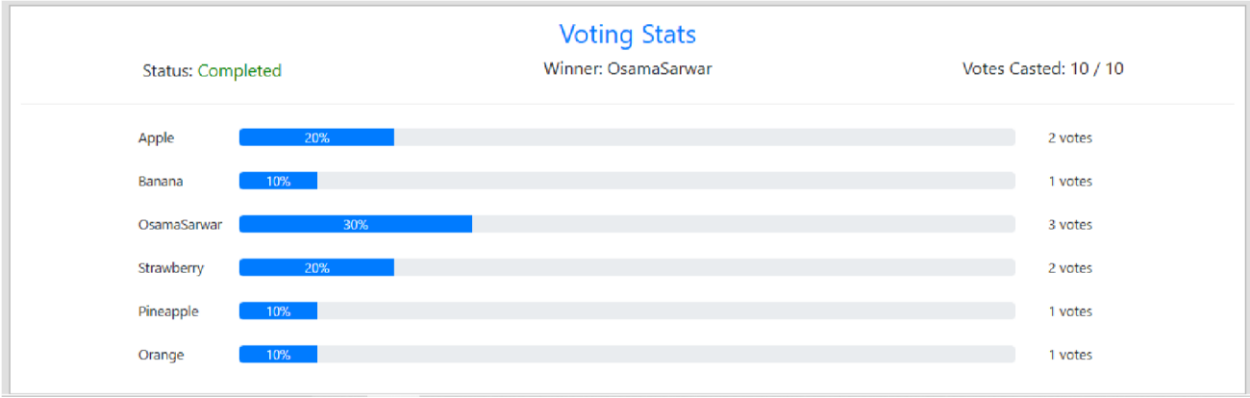


Fig5.2 After final vote

ReactJS: (Code)

```
if (parseInt(voteCount,0)>=parseInt(voteThresh,0)){
   _winner = await simpstorage.methods.checkWinner().call();


}
```

As soon as voteCount becomes equal to voteThreshold (in this case 10), checkWinner method of contract is called and result is returned and displayed.