Change.org - Petition the change you want to see

Sumit Kumar (skumar34@buffalo.edu)
Vibhav Virendra Yawalkar (vibhavvi@buffalo.edu)

Abstract

A platform where people start petitions for local, national or international causes pertaining to environment, healthcare, social, civic and political domains and ask support from people all over the world. People show support by signing these petitions. Participants should be above 18 years and should belong to the same geographical location for which the petition is raised. There's a limit on the number of petitions a person can raise and the person needs to register through an authenticator to verify his age and the geographical location. Domain experts and other qualified participants have a vote weighing higher that of a normal participant. People can also donate for a petition. Based on the number of supporters, decision makers like municipalities, governments and international organizations engage and respond to the petition and take appropriate action.

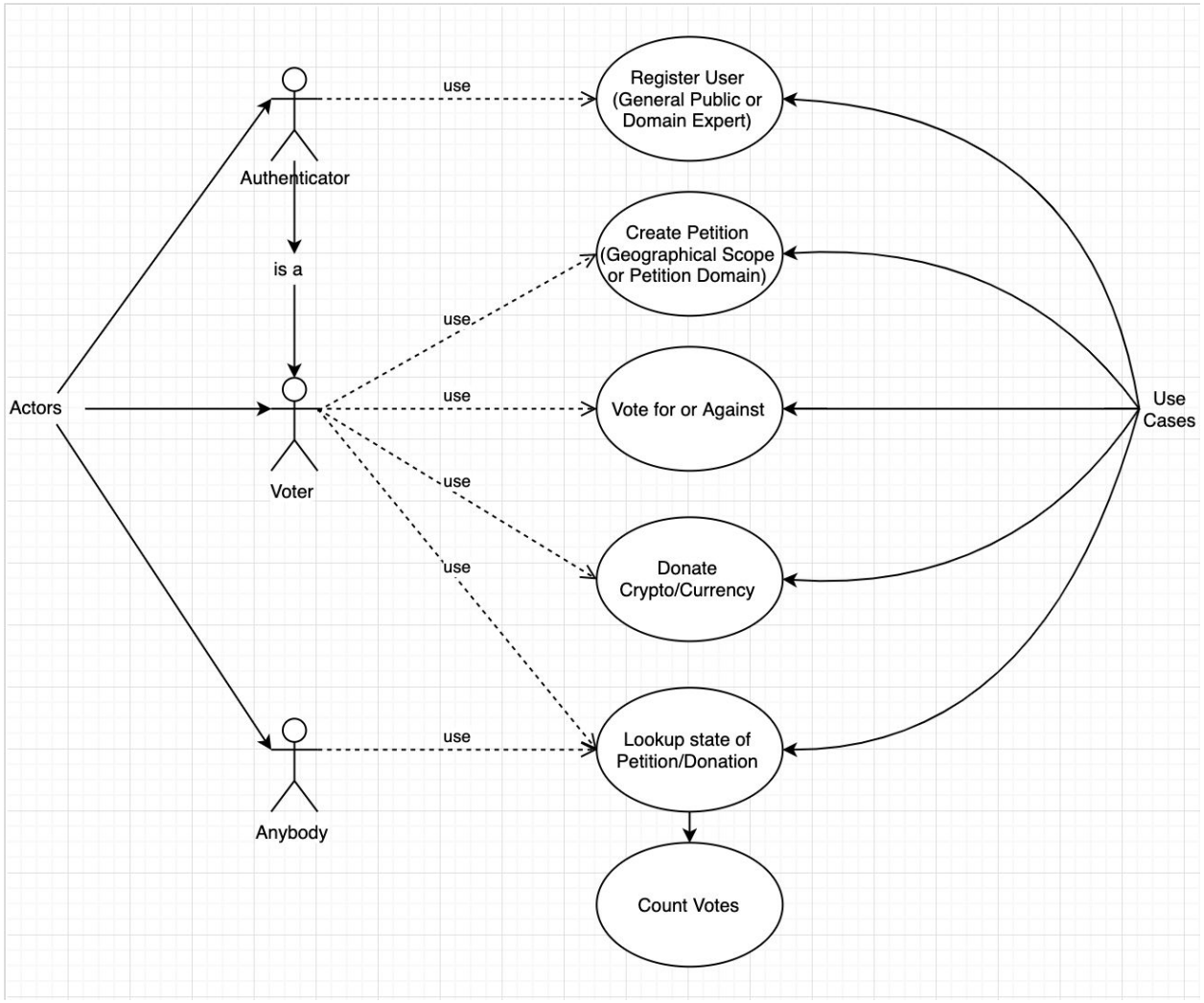
Petition Domains: Environment, Healthcare, Social, Civic, Political.
Petition Scope: Local, National, International.

Rules:

1. Only the chairperson can register a voter
2. Only the chairperson and a registered voter can raise a petition
3. No voter can vote twice on the same petition
4. A voter can vote “for” or “against” the petition
5. A registered voter must be above 18 years of age, to be able to vote.
6. A voter with a broader scope (International scope) can vote for petitions with any scope(Local, National and International)
7. A voter with a narrow scope cannot vote for a petition of broader scope. For example: A voter with a local scope cannot vote for an international petition.
8. There is a limit on the number of petitions a voter can raise.
9. A voter has a limited number of votes.
10. An expert voter has more weightage (5) to his vote than the normal voter weightage (1). A chairperson has weight = 3 for his vote.
11. Anyone i.e chairperson, registered voter or any other person can check the petition status and donation amount for the petition.
12. After voting, in the donation phase only a registered voter can donate an amount > 0 and in multiple of $10.
13. If the difference between the 'for' and 'against' votes to a petition is less than 1, then it is considered as a tie and assert is called.

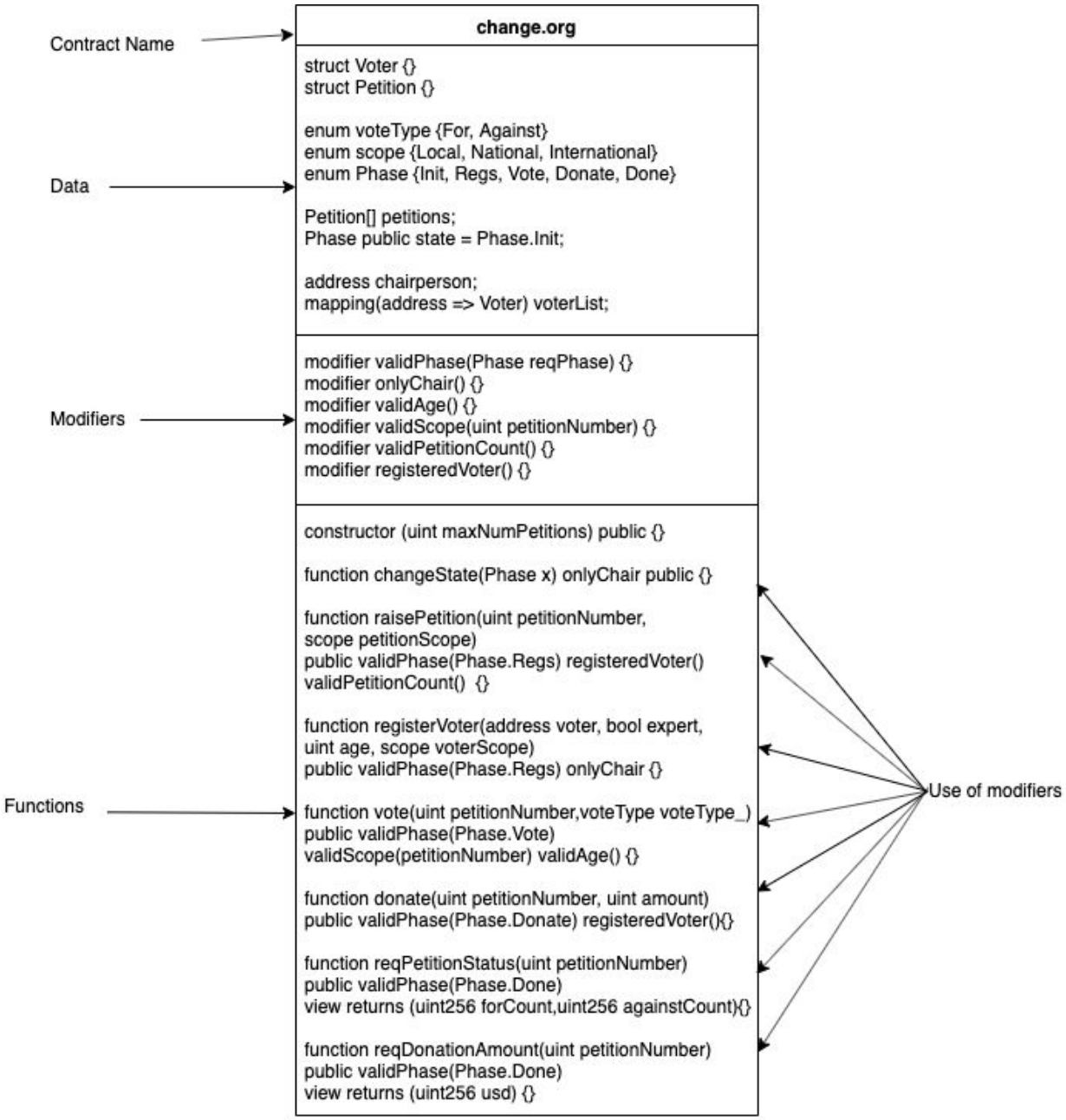
Change.org - Petition the change you want to see

Use Case Diagram



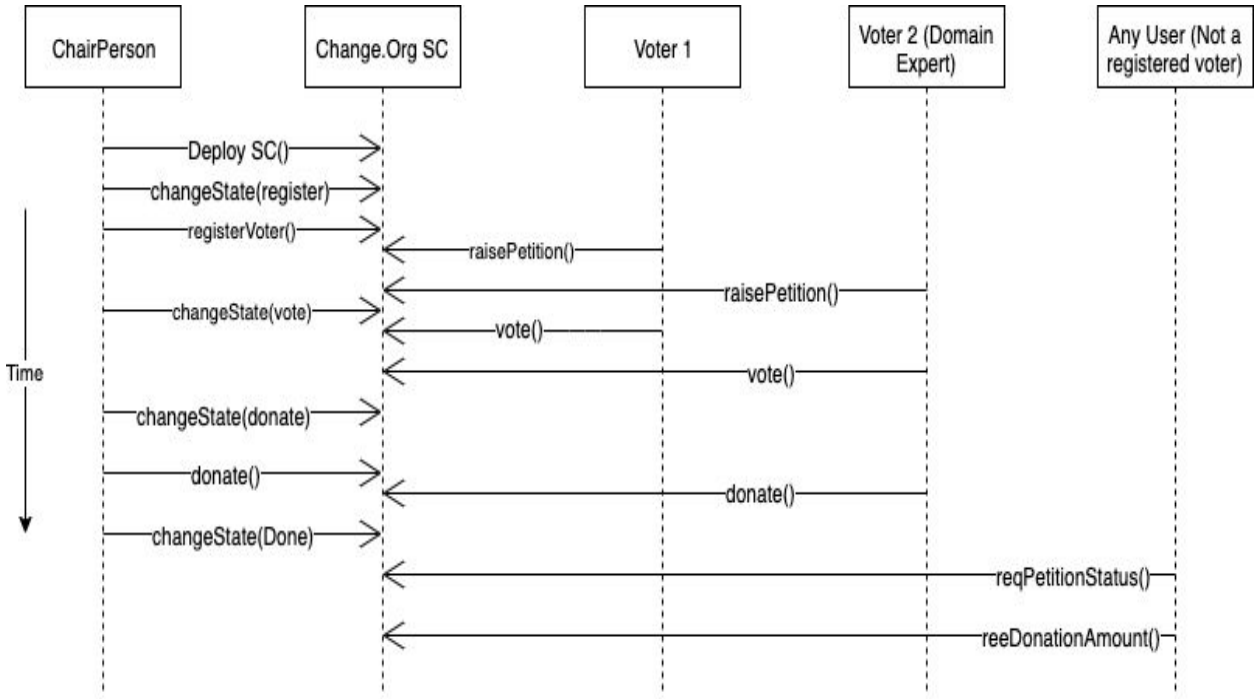
Change.org - Petition the change you want to see

Contract Diagram



Change.org - Petition the change you want to see

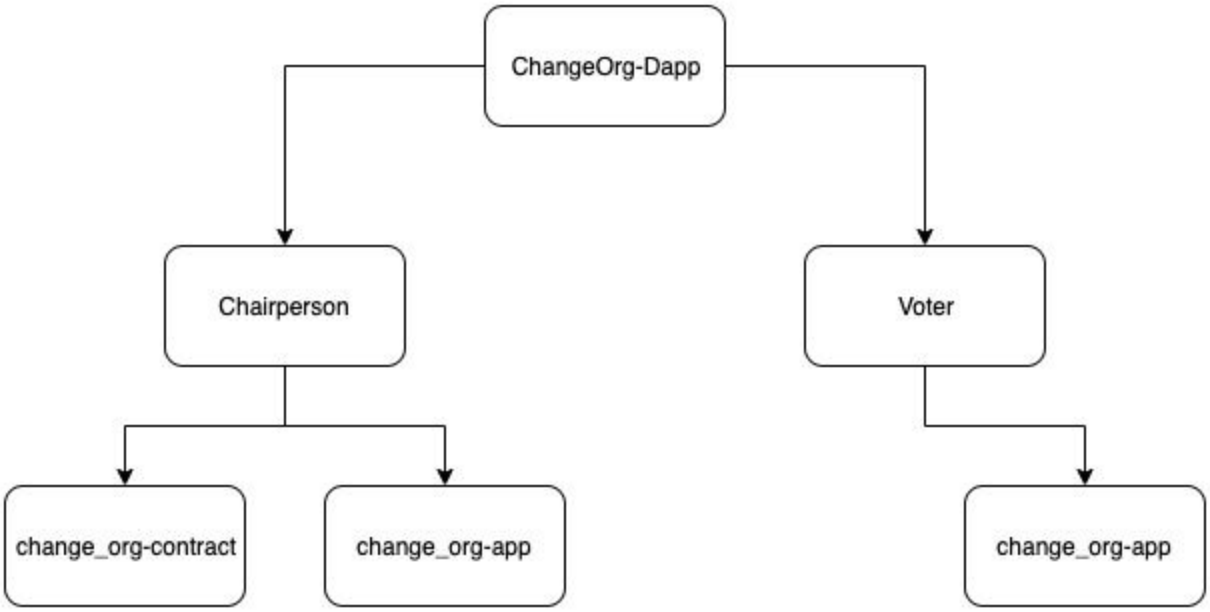
Sequence Diagram



Sequence Diagram

Change.org - Petition the change you want to see

Architecture Diagram



Change.org - Petition the change you want to see

Smart Contract Code

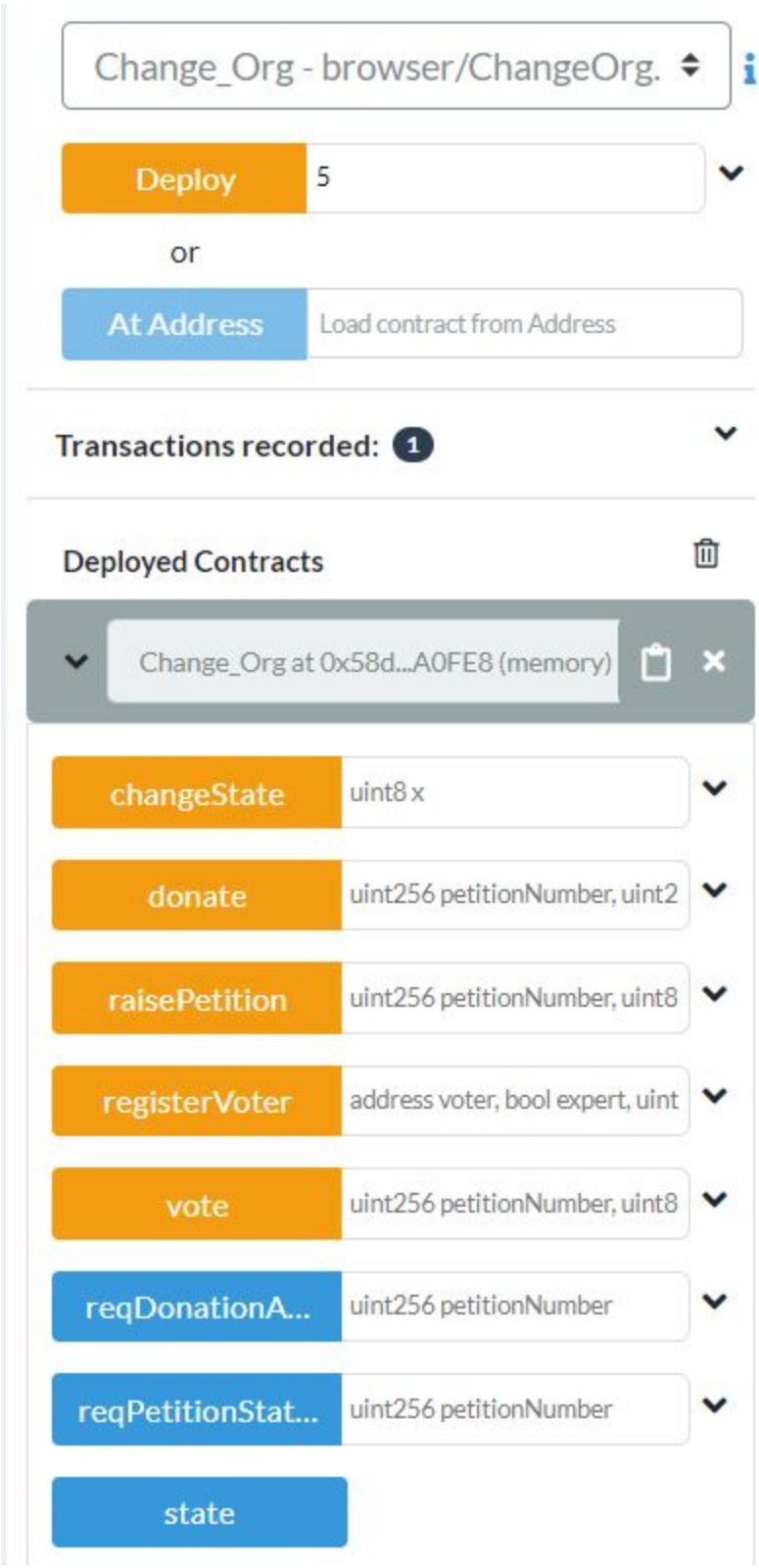
```
pragma solidity ^0.5.2;

contract Change_Org {
    enum voteType {For, Against}
    enum scope {Local, National, International}

    struct Voter {
        uint weight;
        uint voteCount;
        uint petitionCount;
        uint age;
        bool expert; // bool expert if the voter is the domain expert
        scope voterScope;
        bool voted;
        bool registered;
    }

    struct Petition {
        uint256 forCount;
        uint256 againstCount;
        uint donation;
        uint petitionNumber;
        scope petitionScope;
        mapping(address => Voter) petitionVotes;
    }

    Petition[] petitions;

    address chairperson;
    mapping(address => Voter) voterList;

    enum Phase{Init, Regs, Vote, Donate, Done}

    Phase public state = Phase.Init;

    modifier validPhase(Phase reqPhase) {
        require(state == reqPhase);
        _;
    }


modifier onlyChair() {
```

```
        require(msg.sender == chairperson);
        _;
    }

    modifier validAge() {
        require(voterList[msg.sender].age > 18);
        _;
    }

    modifier validScope(uint petitionNumber) {
        require(voterList[msg.sender].voterScope >= petitions[petitionNumber].petitionScope);
        _;
    }

    modifier validPetitionCount() {
        require(voterList[msg.sender].petitionCount > 0);
        _;
    }

    modifier registeredVoter() {
        require(voterList[msg.sender].registered == true);
        _;
    }

    constructor (uint maxNumPetitions) public {
        chairperson = msg.sender;
        voterList[chairperson].weight = 2; //weight 2 for chairperson
        voterList[chairperson].voteCount = 5;
        voterList[chairperson].petitionCount = 100;
        voterList[chairperson].age = 35;
        voterList[chairperson].voterScope = scope.International;
        voterList[chairperson].registered = true;
        petitions.length = maxNumPetitions;
    }

    function changeState(Phase x) onlyChair public {
        require (x > state );
        state = x;
    }

    //The chairperson or any registered voter with a valid petition count can raise a petition in a
Registration phase
```

```
    function raisePetition(uint petitionNumber, scope petitionScope) public
validPhase(Phase.Regs) registeredVoter() validPetitionCount()
{
        require(petitionNumber >= 0); // valid petition number
        require(petitionScope <= scope.International); // valid petition type

        voterList[msg.sender].petitionCount = voterList[msg.sender].petitionCount - 1;
        petitions[petitionNumber].petitionScope = petitionScope;

        petitions[petitionNumber].forCount = petitions[petitionNumber].againstCount =
petitions[petitionNumber].donation = 0;
    }

    //Function to register a voter, Only chair can register a voter with his age, scope and expert
flag
    function registerVoter(address voter, bool expert, uint age, scope voterScope) public
validPhase(Phase.Regs) onlyChair
{
        voterList[voter].registered = true;
        voterList[voter].age = age;
        voterList[voter].voterScope = voterScope;
        if (true == expert)
            voterList[voter].weight = 3; // weight = 3 for the domain expert
        else
            voterList[voter].weight = 1;

        voterList[voter].voteCount = 5;
        voterList[voter].petitionCount = 3;
    }

    // Function to vote for a particular petition
    function vote(uint petitionNumber, voteType voteType_) public validPhase(Phase.Vote)
validScope(petitionNumber) validAge()
{
        require(petitions[petitionNumber].petitionVotes[msg.sender].voted == false); // meaning the
voter shouldn't have voted for the same proposal before
        require (voterList[msg.sender].voteCount > 0);
        require (petitionNumber < petitions.length);
        //require(voterList[msg.sender].voterScope >= petitions[petitionNumber].petitionScope);

        voterList[msg.sender].voteCount = voterList[msg.sender].voteCount - 1;
        if(voteType_ == voteType.For) {
            petitions[petitionNumber].forCount += voterList[msg.sender].weight;
```

```
        } else { // _type == voteType.Against
            petitions[petitionNumber].againstCount += voterList[msg.sender].weight;
        }
        petitions[petitionNumber].petitionVotes[msg.sender].voted = true;
    }

    // Function to donate for a particular petition, after voting
function donate(uint petitionNumber, uint amount) public validPhase(Phase.Donate)
registeredVoter() {
        assert(amount > 0 && amount % 10 == 0); // donation amount should be greater than 0
and in multiple of $10
        petitions[petitionNumber].donation += amount;
    }

    // Function to get the final status of a particular petition after the voting phase is done.
    function reqPetitionStatus(uint petitionNumber) public validPhase(Phase.Done) view returns
(uint256 forCount, uint256 againstCount) {
        forCount = petitions[petitionNumber].forCount;
        againstCount = petitions[petitionNumber].againstCount;
        if(forCount > againstCount)
            assert(forCount - againstCount > 1); // case where we have a tie
        else
            assert(againstCount - forCount > 1); // case where we have a tie
    }

    // Function to get the total donation amount for a particular petition after the donation phase is
done.
    function reqDonationAmount(uint petitionNumber) public validPhase(Phase.Done) view
returns (uint256 usd) {
        usd = petitions[petitionNumber].donation;
    }

function getCurrentState() public view returns (Phase currentState){
        currentState = state;
    }
}
```

Change.org - Petition the change you want to see



App.js file

```
App = {
  web3Provider: null,
  contracts: {},
  voterScope: {},
  names: new Array(),
  url: 'http://127.0.0.1:7545',
  chairPerson:null,
  currentAccount:null,
  statesName: {
    0: "Init",
    1: "Register",
    2: "Vote",
    3: "Donate",
    4: "Done"
  },
  petitionScopeDict: {
    0: "Local",
    1: "National",
    2: "International"
  },
  currentPhase: 0,
  init: function() {
    $.getJSON('../proposals.json', function(data) {
      var proposalsRow = $('#proposalsRow');
      var proposalTemplate = $('#proposalTemplate');

      for (i = 0; i < data.length; i ++) {
        proposalTemplate.find('.panel-title').text(data[i].name + " : " +
App.petitionScopeDict[data[i].scope]);
        proposalTemplate.find('.panel-link').attr('href', data[i].link);
        proposalTemplate.find('img').attr('src', data[i].picture);
        proposalTemplate.find('img').attr('title', data[i].tooltip);
        proposalTemplate.find('.btn-vote-for').attr('data-id', data[i].for_id);
        proposalTemplate.find('.btn-vote-for').attr('petitionScope', data[i].scope);
        proposalTemplate.find('.btn-vote-against').attr('data-id', data[i].against_id);
        proposalTemplate.find('.btn-vote-against').attr('petitionScope', data[i].scope);
        proposalTemplate.find('.btn-donate').attr('data-id', data[i].donate_id);
        proposalTemplate.find('.donate-amt').attr('data-id', data[i].donate_amt_id);
        proposalTemplate.find('.register-petition').attr('petitionNumber', data[i].petition_id);
        proposalTemplate.find('.register-petition').attr('id', "registerPetition" + data[i].petition_id);
        proposalTemplate.find('.register-petition').attr('petitionScope', data[i].scope);
        proposalTemplate.find('.petition-status').attr('petitionNumber', data[i].petition_id);
```

```
      proposalTemplate.find('.petition-status').attr('petitionScope', data[i].scope);
      proposalTemplate.find('.btn-req-donation-amt').attr('petitionNumber', data[i].petition_id);
      proposalTemplate.find('.btn-req-donation-amt').attr('petitionScope', data[i].scope);
      proposalTemplate.find('.for-count-span').attr('id', "forCountSpan" + data[i].petition_id);
      proposalTemplate.find('.against-count-span').attr('id', "againstCountSpan" +
data[i].petition_id);



      proposalsRow.append(proposalTemplate.html());
      App.names.push(data[i].name);
    }
  });
  return App.initWeb3();
 },

 initWeb3: function() {
  // Is there is an injected web3 instance?
  if (typeof web3 !== 'undefined') {
   App.web3Provider = web3.currentProvider;
  } else {
   // If no injected web3 instance is detected, fallback to the TestRPC
   App.web3Provider = new Web3.providers.HttpProvider(App.url);
  }
  web3 = new Web3(App.web3Provider);

  ethereum.enable();

  App.populateAddress();
  return App.initContract();
 },

 initContract: function() {
  $.getJSON('Change_Org.json', function(data) {
   // Get the necessary contract artifact file and instantiate it with truffle-contract
   var voteArtifact = data;
   App.contracts.vote = TruffleContract(voteArtifact);

   // Set the provider for our contract
   App.contracts.vote.setProvider(App.web3Provider);

   App.getChairperson();
   return App.bindEvents();
  });
```

```
  },

  bindEvents: function() {
    $(document).on('click', '.btn-vote-for', App.handleVote);
    $(document).on('click', '.btn-vote-against', App.handleVoteAgainst);
    $(document).on('click', '.btn-donate', App.handleDonate);
    $(document).on('click', '#register', function(){ var ad = $('#enterAddress').val();
App.handleRegister(ad); });
    $(document).on('click', '#changeStateBtn', function(){ var newState =
$('#enterStateOpt').val(); App.handleChangeState(newState);});
    $(document).on('click', '.register-petition', App.handleRaisePetition);
    $(document).on('click', '.petition-status', App.handlePetitionStatus);
    $(document).on('click', '#currentStateBtn', App.handleCurrentState);
    $(document).on('click', '.btn-req-donation-amt', App.handleRequestDonationAmount);
  },

  populateAddress : function(){
    new Web3(new Web3.providers.HttpProvider(App.url)).eth.getAccounts((err, accounts) => {
      jQuery.each(accounts,function(i){
        if(web3.eth.coinbase != accounts[i]){
          var optionElement = '<option value="'+accounts[i]+'">'+accounts[i]+'</option';
          jQuery('#enterAddress').append(optionElement);
        }
      });
    });
  },

  getChairperson : function(){
    App.contracts.vote.deployed().then(function(instance) {
      return instance;
    }).then(function(result) {
      App.chairPerson = result.constructor.currentProvider.selectedAddress.toString();
      App.currentAccount = web3.eth.coinbase;

      if(App.chairPerson != App.currentAccount){
        jQuery('#addressDiv').css('display','none');
        jQuery('#registerDiv').css('display','none');
      }else{
        jQuery('#addressDiv').css('display','block');
        jQuery('#registerDiv').css('display','block');
      }
    });
  },
```

```
  handleRegister: function(addr){
    alert("Registering " + addr);
    var voteInstance;
    App.contracts.vote.deployed().then(function(instance) {
      voteInstance = instance;

      var expert = $('#isExpert').val();
      var age = $('#ageId').val();
      var voter_scope = parseInt($('#voterScopeVal').val());

      return voteInstance.registerVoter(addr, expert, age, voter_scope);
    }).then(function(result){
      if(result){
        var voter_scope = $('#voterScopeVal').val();
        App.voterScope[addr] = parseInt(voter_scope);

        alert("Registration done successfully for " + addr);
      }
    }).catch(function(err){
      alert("Registration of " + addr + " not done successfully due to revert");
      // alert("Something Went Wrong. See Ganache logs. " + err['code']);
    });
  },

  handleChangeState: function(newState){
    alert("Changing Phase to " + App.statesName[newState]);
    var voteInstance;
    App.contracts.vote.deployed().then(function(instance) {
      voteInstance = instance;

      return voteInstance.changeState(newState);
    }).then(function(result){
      if(result){
        App.currentPhase = newState;
        alert("Phase changed successfully to " + App.statesName[newState]);
      }
    }).catch(function(err){
      alert("Phase change to " + App.statesName[newState] + " unsuccessful due to revert");
      // alert("Something Went Wrong. See Ganache logs. " + err['code']);
    });
  },
```

```
handleCurrentState: function(event){
  event.preventDefault();
  var voteInstance;

  web3.eth.getAccounts(function(error, accounts) {
    var account = accounts[0];

    App.contracts.vote.deployed().then(function(instance) {
      voteInstance = instance;

      return voteInstance.getCurrentState();
    }).then(function(result){
      if(result){
        App.currentPhase = result;
        alert("Current Phase: " + App.statesName[result]);
      }
    }).catch(function(err){
      alert("Current Phase fetching failed!");
      // alert("Something Went Wrong. See Ganache logs. " + err['code']);
    });
  });
},

handleVote: function(event) {
  event.preventDefault();

  var proposalId = parseInt($(event.target).data('id'));
  var petitionScope = parseInt($(event.target).attr('petitionScope'));
  var voteInstance;

  web3.eth.getAccounts(function(error, accounts) {
    var account = accounts[0];

    App.contracts.vote.deployed().then(function(instance) {
      voteInstance = instance;

      return voteInstance.vote(proposalId, 0, {from: account});
    }).then(function(result){
      if(result){
        alert(account + " voting successful");
      }
    }).catch(function(err){
      alert(account + " voting unsuccessful due to revert");
```

```
      // alert("Something Went Wrong. See Ganache logs. " + err['code']);
    });
  });
},

handleVoteAgainst: function(event) {
  event.preventDefault();

  var proposalId = parseInt($(event.target).data('id'));
  var petitionScope = parseInt($(event.target).attr('petitionScope'));
  var voteInstance;
  proposalId = proposalId % 4;

  web3.eth.getAccounts(function(error, accounts) {
    var account = accounts[0];

    App.contracts.vote.deployed().then(function(instance) {
      voteInstance = instance;

      return voteInstance.vote(proposalId, 1, {from: account});
    }).then(function(result){
      if(result){
        alert(account + " vote against done successfully");
      }
    }).catch(function(err){
      alert(account + " vote against unsuccessful due to revert");
      // alert("Something Went Wrong. See Ganache logs. " + err['code']);
    });
  });
},

handleDonate: function(event) {
  event.preventDefault();

  var proposalId = parseInt($(event.target).data('id'));
  var amount = $(".donate-amt[data-id=" + (proposalId+4) +"]").val();
  var voteInstance;

  proposalId = proposalId % 4;

  web3.eth.getAccounts(function(error, accounts) {
    var account = accounts[0];
    App.contracts.vote.deployed().then(function(instance) {
```

```
      voteInstance = instance;

      return voteInstance.donate(proposalId, amount, {from: account});
    }).then(function(result){
      if(result){
        alert(account + " donation successful");
      }
    }).catch(function(err){
      if(App.currentPhase < 3){
        alert("Invalid Phase for this action");
      }else if(App.voterScope[account] === undefined){
        alert("Not a Registered Voter");
      }else{
        alert(account + " donation unsuccessful due to revert");
      }
      // alert("Something Went Wrong. See Ganache logs. " + err['code']);
    });
  });
},

handleRaisePetition: function(event) {
  event.preventDefault();

  var petitionNumber = parseInt($(event.target).attr('petitionNumber'));
  var petitionScope = parseInt($(event.target).attr('petitionScope'));

  var voteInstance;

  web3.eth.getAccounts(function(error, accounts) {
    var account = accounts[0];

    App.contracts.vote.deployed().then(function(instance) {
      voteInstance = instance;

      return voteInstance.raisePetition(petitionNumber, petitionScope, {from: account});

    }).then(function(result){
      if(result){
        alert(App.petitionScopeDict[petitionScope] + " Petition: " + petitionNumber + " raised
successful");
        $("#registerPetition" + petitionNumber).attr("disabled","disabled");
      }
    }).catch(function(err){
```

```
      alert(App.petitionScopeDict[petitionScope] + " Petition: " + petitionNumber + " unsuccessful
due to revert");
      // alert("Something Went Wrong. See Ganache logs. " + err['code']);
    });
  });
 },

 handlePetitionStatus : function(event) {
  event.preventDefault();

  var petitionNumber = parseInt($(event.target).attr('petitionNumber'));
  alert("Status of Petition: " + petitionNumber);
  var voteInstance;

  App.contracts.vote.deployed().then(function(instance) {
   voteInstance = instance;
   return voteInstance.reqPetitionStatus(petitionNumber);
  }).then(function(res){
   // alert("Petition: " + petitionNumber + " forCount: " + res[0].c[0] + " " + " againstCount: " +
res[1].c[0]);
   $("#forCountSpan"+petitionNumber).text("For Count: " + res[0].c[0]);
   $("#againstCountSpan"+petitionNumber).text("Against Count: " + res[1].c[0]);
  }).catch(function(err){
   if(App.currentPhase < 4){
    alert("Invalid Phase for this action");
   }else{
    alert("Something Went Wrong. See Ganache logs. " + err['code']);
   }
  });
 },

 handleRequestDonationAmount : function(event) {
  event.preventDefault();

  var petitionNumber = parseInt($(event.target).attr('petitionNumber'));

  alert("Fetching Donation Amount of Petition: " + petitionNumber);

  var voteInstance;

  App.contracts.vote.deployed().then(function(instance) {
   voteInstance = instance;
   return voteInstance.reqDonationAmount(petitionNumber);
```

```
    }).then(function(res){
      alert("Donation Amount of Petition: " + petitionNumber + " is " + res.c[0]);
    }).catch(function(err){
      alert("Invalid Phase for this action");
      // alert("Something Went Wrong. See Ganache logs. " + err['code']);
    });
  }
};

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

Change.org - Petition the change you want to see

Remix Testing Plan:

Initial setup:

1. Check the state initially, once the contract is deployed by the chairperson, we are in the init phase[0]. Now change the state to Registration state [1], in this stage the chairperson can register the voters by calling the registerVoter function with the voter address, age, scope and expert flag. Set the expert flag if the voter is a domain expert. Add one domain expert voter with international scope[2] and one general voter with local scope[0]. Thus we have three voters chairperson, domain expert and general voter. In the registration phase, the registered voters can raise petitions by calling the raisePetition() function. Raise 2 petitions, one by using the chairperson address and other using the domain expert address. Petition Scope can be set as Local [0], and International[2] for each of the petition.'

   Voters:

   Chairperson - International scope[2]
   Domain Expert Voter - International scope[2]
   General Voter - Local scope [0]

   Petitions:

   Petition 1 - Local scope[0] (raised by the chairperson)
   Petition 2 - International scope[2] (raised by the domain expert)

Positive Test case/flow:

1. Change the state to voting state using the chairperson address. Chairperson calls the vote function to vote 'for' petition 1 and vote 'against' petition 2. Change address to domain expert, the expert votes 'for' petition 1 and 'against' petition 2. Change the address to general voter and the voter can vote only for petition 1 as his scope is local. He votes in 'for' of petition 1.
2. The chairperson changes the state to donate stage, where the chair donated $1000 for petition 1 and expert donated $500 for petition 1. General voter didn't donate.
3. Any other address (user) calls reqPetitionStatus() for petition 1 to get the status of petition and gets the 'for' vote counts and 'against' vote counts.
4. General voter calls the reqDonationAmount for petition 1 and views the total donation amount $1500.

Negative Test scenarios:

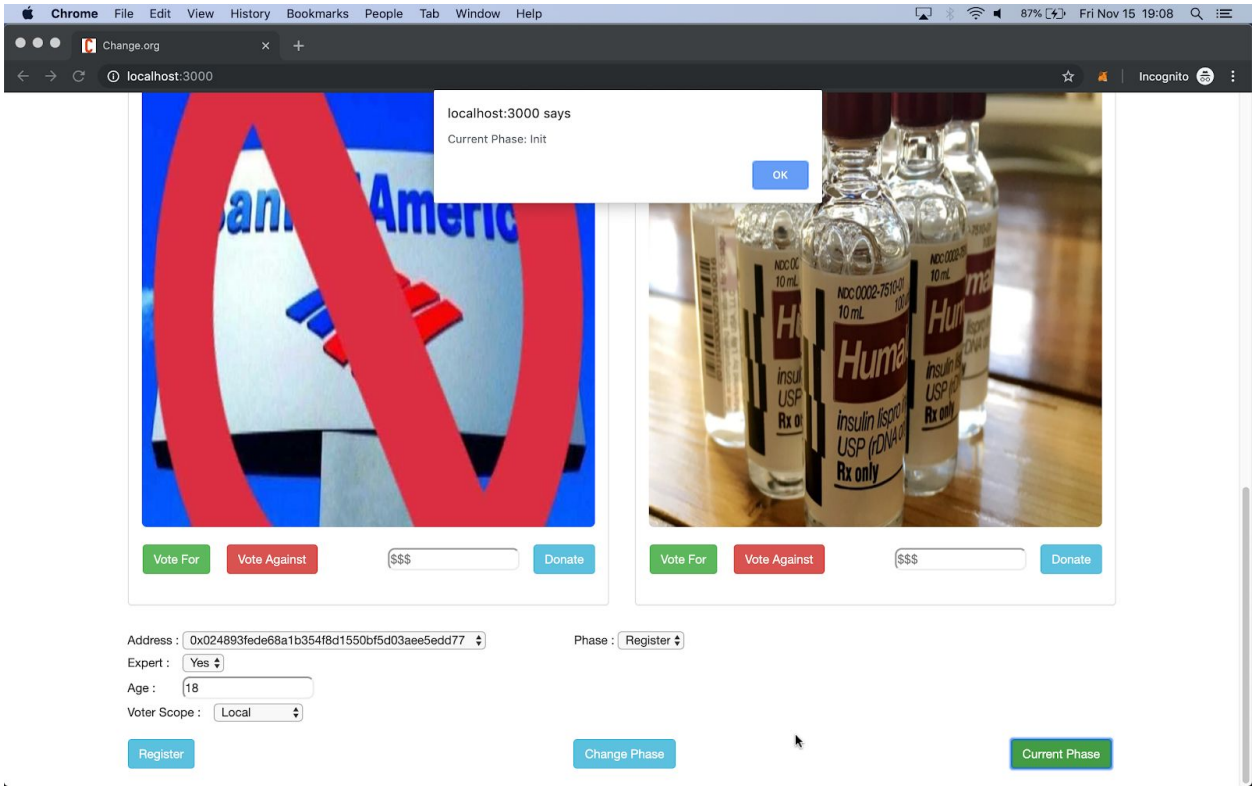
1. After voting once for the petition '1', the expert voter tries to vote again for the same petition during the voting phase, and the transaction is reverted, since any voter can't vote twice for the petition.
2. General voter (local scope) in the above case tries to vote for petition 2, the petition with the international scope. The transaction gets reverted, as a voter with local scope can't vote for a broader scope petition. Modifier validScope causes the revert.

3. Any other address (user account) tries to raise a petition using the raisePetition call. The transaction gets reverted, as only a registered voter can raise a petition. Modifier registeredVoter() causes the revert.

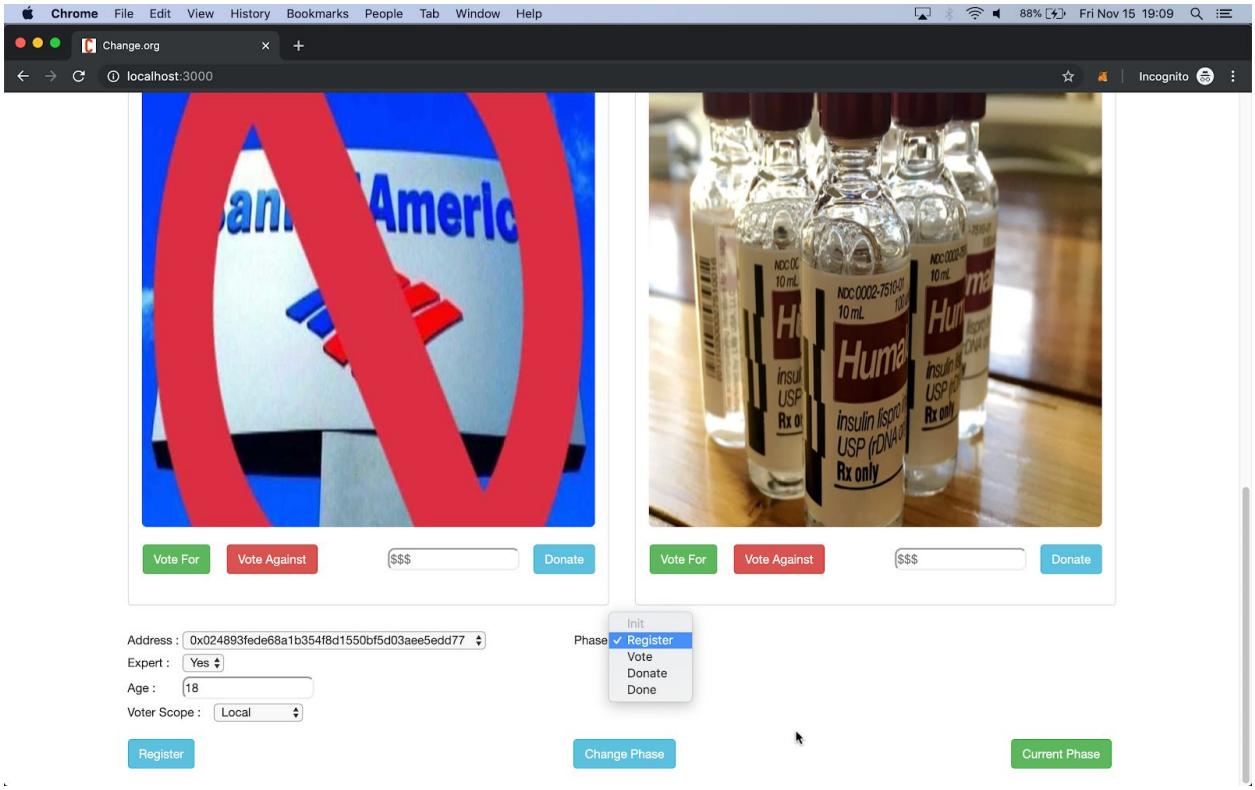
Change.org - Petition the change you want to see

Workflow Instructions with UI Screenshots:

1. First we check the current phase using the <Current Phase> button on the UI. Initially the current phase is Init.

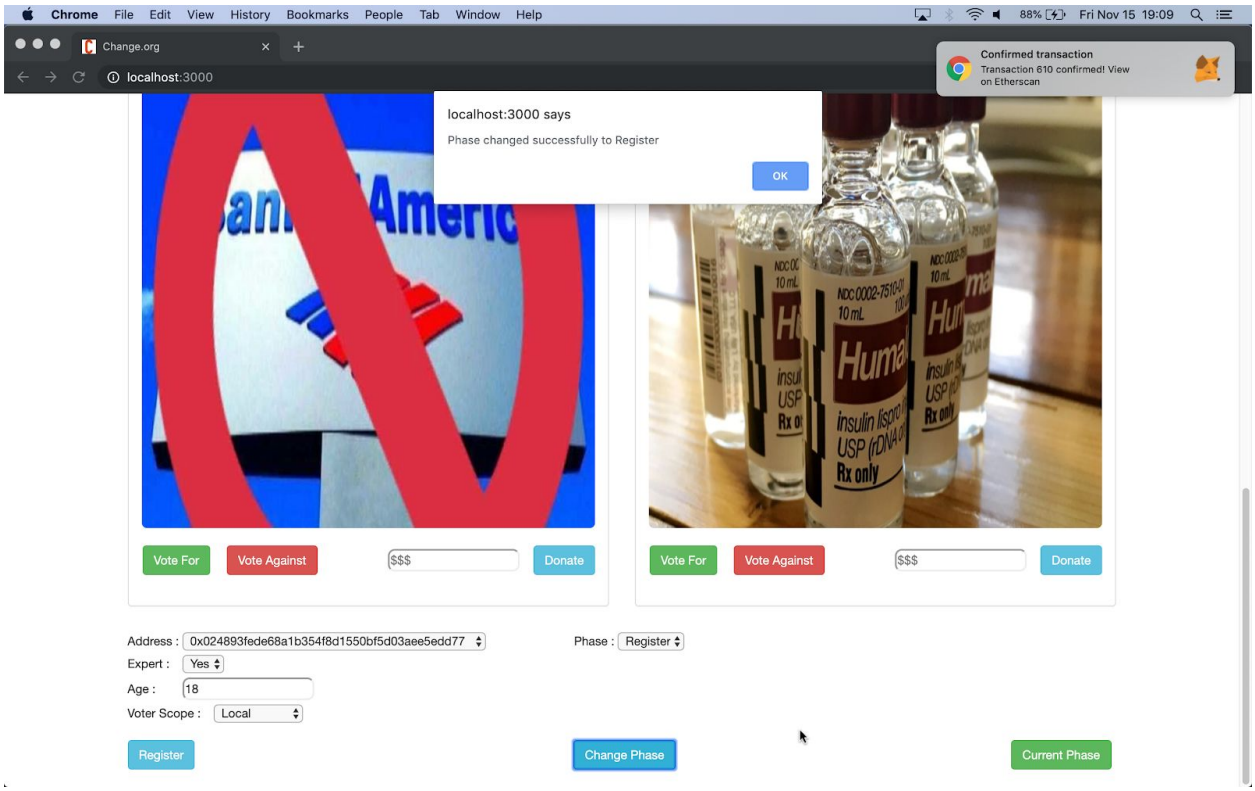


2. Now we select the chairperson account in metamask and select the phase as Register from the Phase drop down menu and press the change phase button.
   Now our phase is changed to the Register phase.

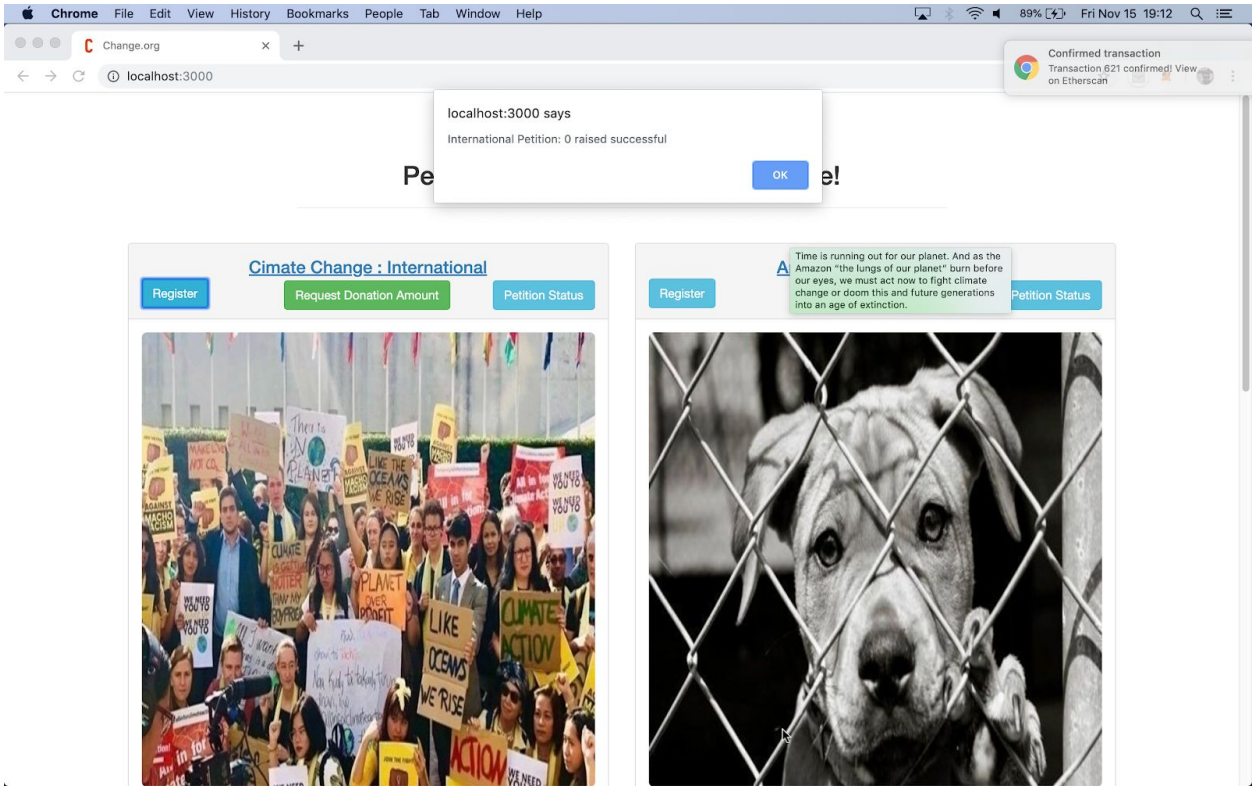


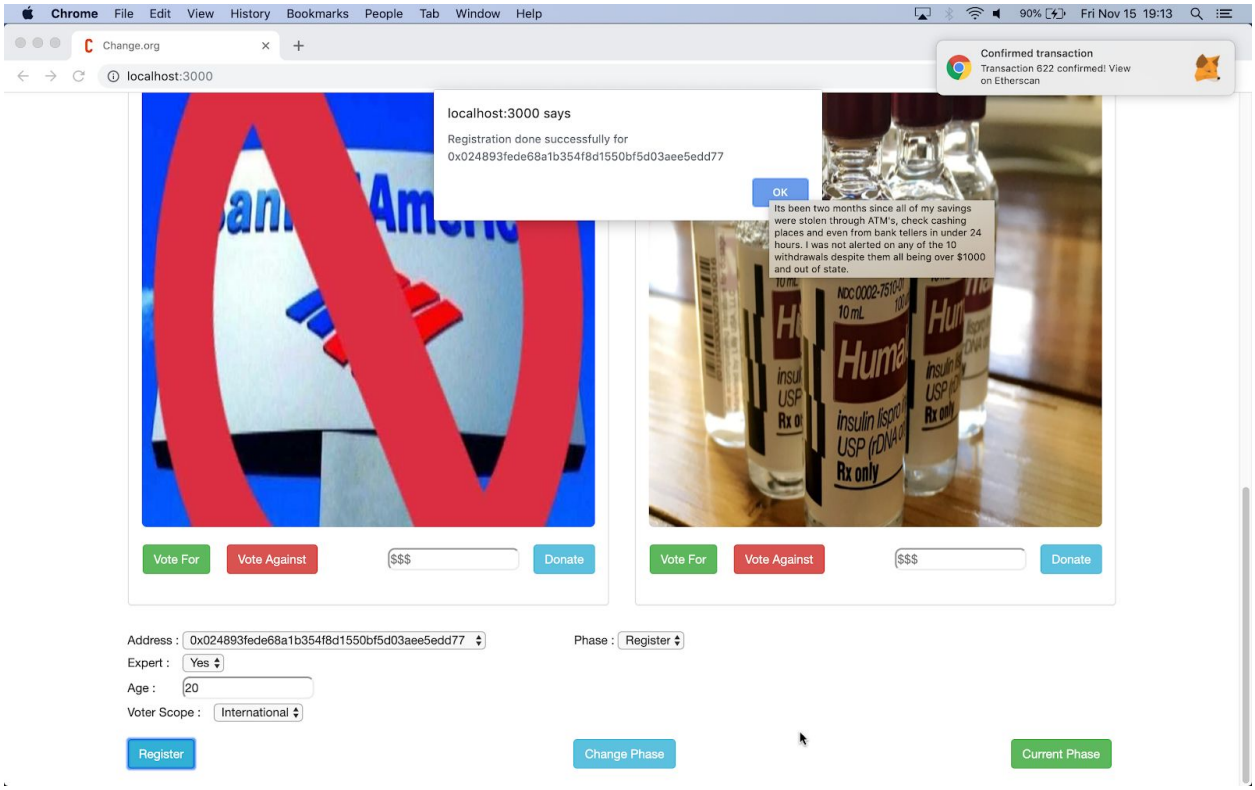
3. In register phase we now can register a voter and the petitions.
   Petitions are displayed on the web page, the chairperson registers them using the register button on the UI. To register the petition, we click on the register button on the petition.The chairperson or any registered voter can register a petition.

   We first register all the petitions displayed on the UI (4 of them).
   Then we register voters by selecting their addresses from the drop down address list.
   We select the attributes of the voters <Expert, Age, Voter Scope> and click on the register button.

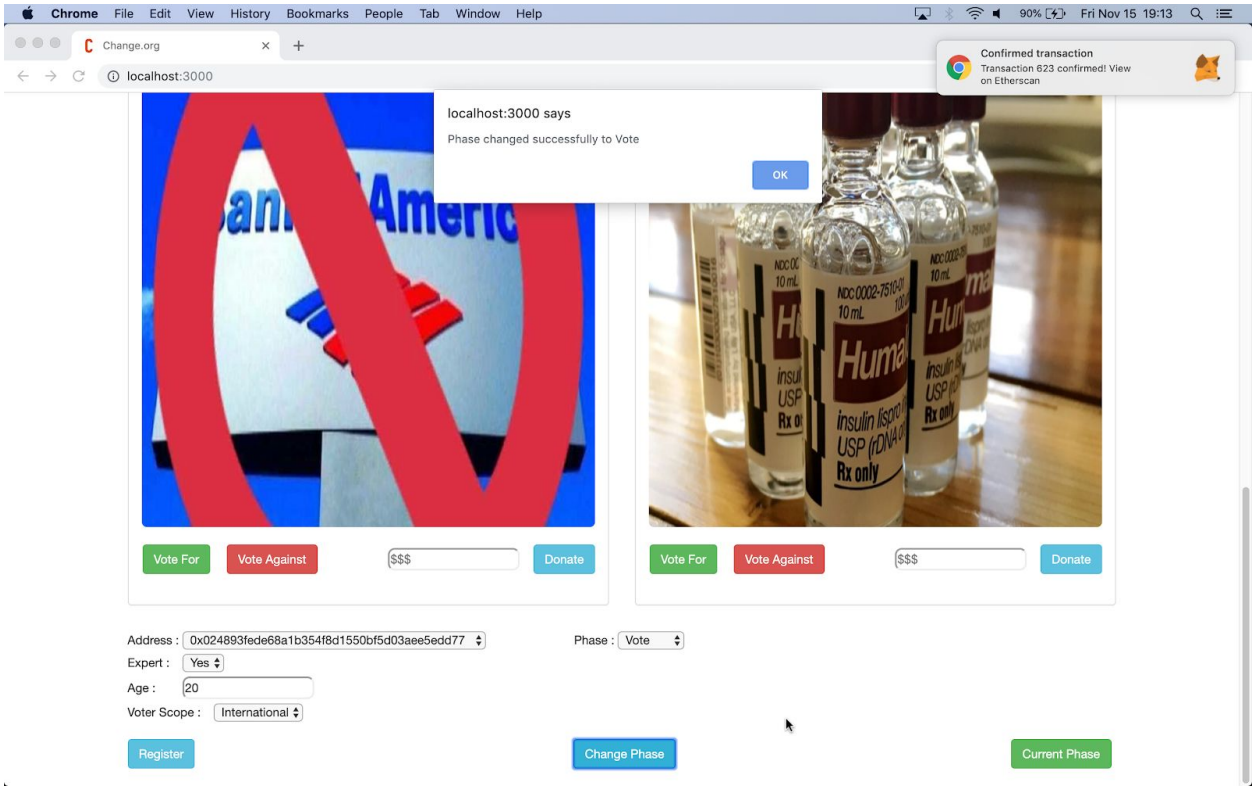


Change.org - Petition the change you want to see

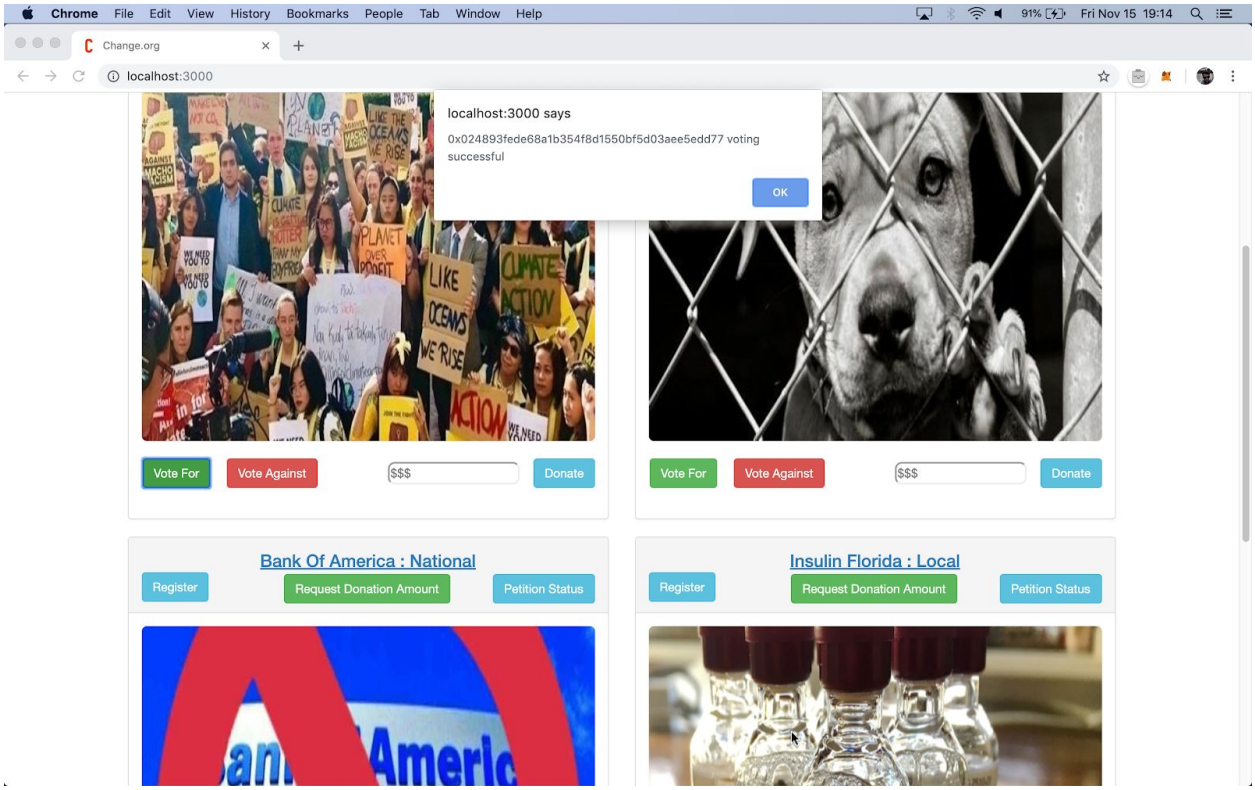




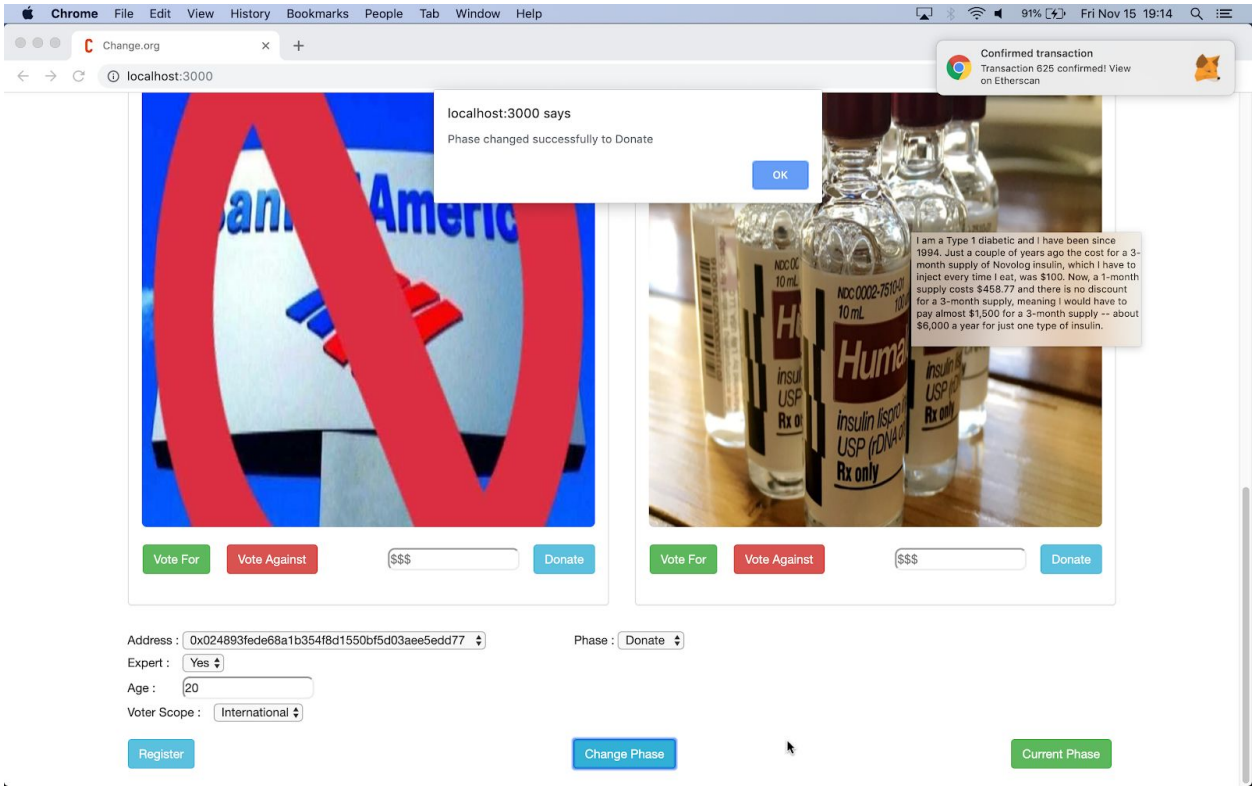
4. Now, we select the Vote phase from the phase drop down list and click on change phase.
   Our phase is now changed to the Vote Phase.



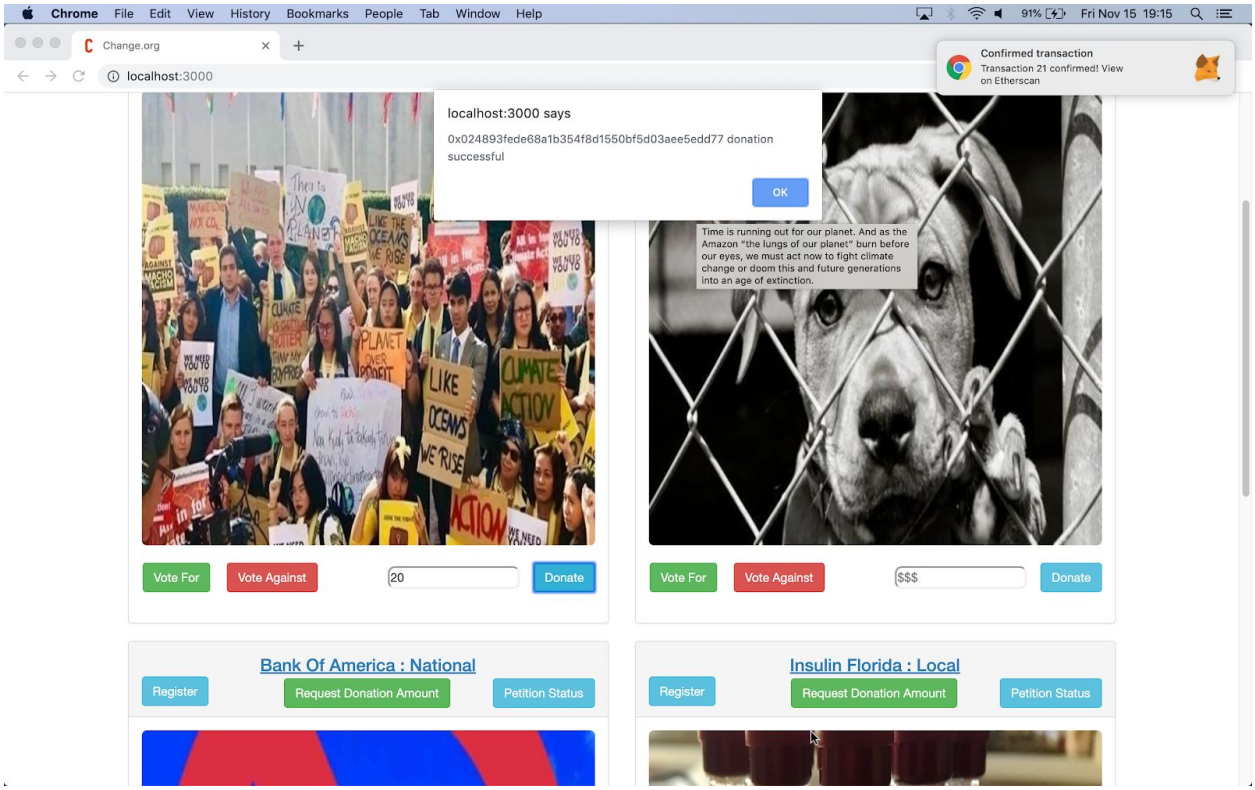
5. We now select the registered user account from MetaMask and click on Vote For or Vote Against for a registered petition.
   The vote is recorded and we can do the voting for multiple registered addresses.



6. Now we select the chairperson address and change the Phase to the Donate Phase, using the phase drop down menu and the change state button. Now the phase is changed to donate phase and we can now donate the amount in dollars using any address from Ropsten, it could be a registered voter or a non-registered person.

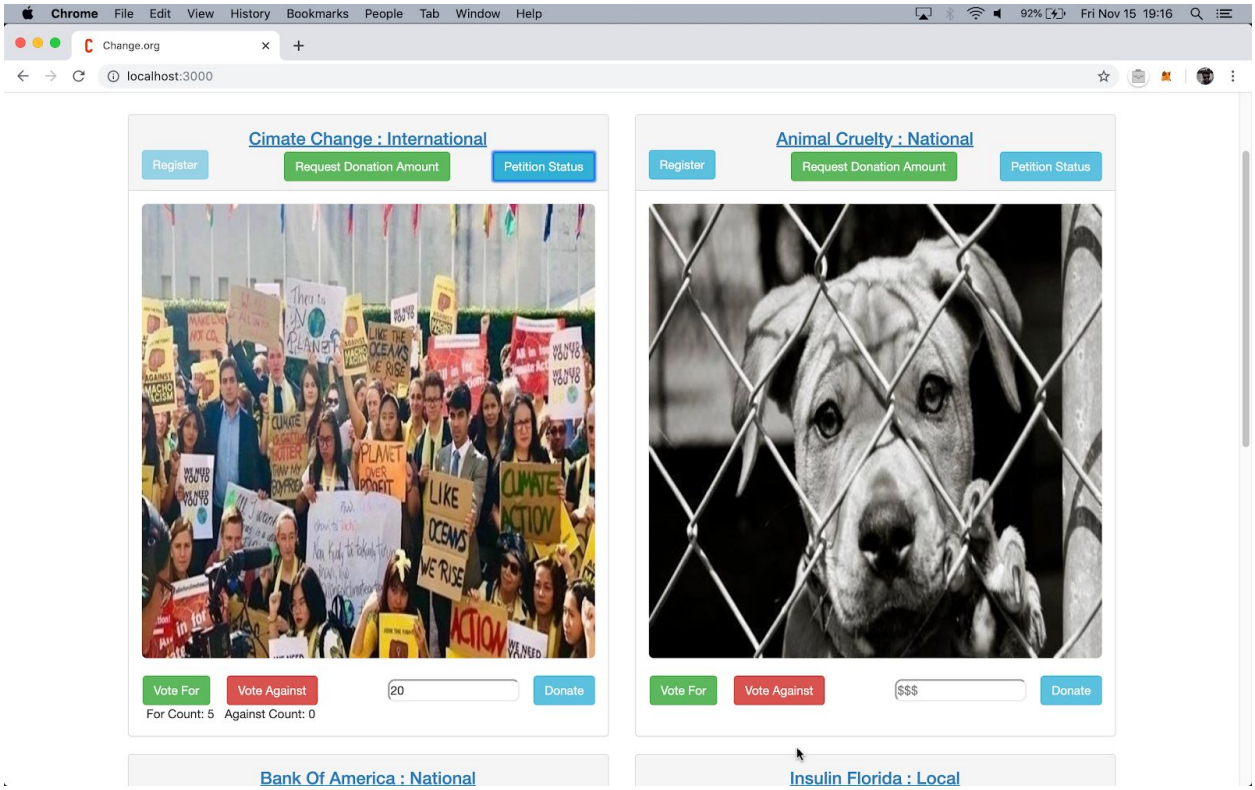


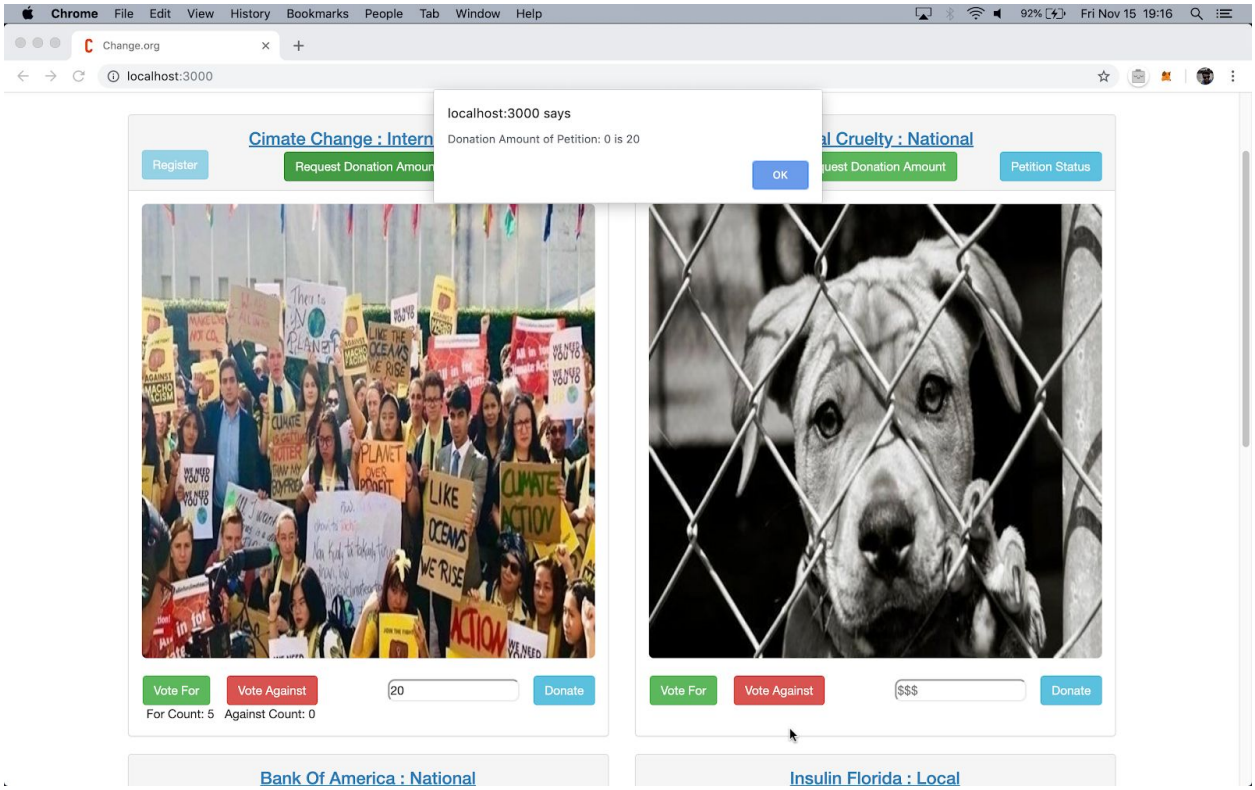
Change.org - Petition the change you want to see



7. We now move to the next phase which is phase Done using the chairperson's address and using the change phase button on the UI.
   Once our phase is Done, any address , registered or non-registered can query the petition status using the PetitionStatus button on the particular petition.
   Once this button is clicked we can get the number of For and Against votes displayed on the UI.
   Similarly we can get the donation amount for a particular petition using any of the addresses(users).
   The donation amount will be displayed in the alert box.

Change.org - Petition the change you want to see





Change.org - Petition the change you want to see