**Testing Gatekeeper and Ballot**

This assumes the DAO has already been instantiated with members.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Step 1 *Deploy the Contracts*

Go to the directory: ~/.eris/apps/SOLIDITY/gatekeeper

Type the command:

eris pkgs do --chain newchain3 --address 59010E90957DFF9DFD8DB95980996463230378E1

Step 2 *Give Gatekeeper Coid Data and run the Javascripts*

Here is formData, as a JSON object. Now, here is a little trick. Notice that in the javascript for gatekeeper, we have the method:

app.post("/gatekeeper", function(req,res){ var formdata = req.body;…

You can comment out “var formdata = req.body “ and put

var formdata = JSON.stringify(obj)

Note that for testing data, use:

var obj =

{

"pubKey": "02d7ceb37a16fde15a5237652b31a52d94def283a2ab09aaf5d1af48db1b84e20a",

"sig": "7051442bbf18bb2c86cbc8951a07e27ec6ba05ac3fa427e4c6b948e3dcf91a94046b048edf52445fb22cc776a94b87c3f55426f993458ec744f61f09fb46eeaa",

"msg": "8836a77b68579d1d8d4427c0cda24960f6c123f17ccf751328cc621d6237da22",

"uniqueId": "83D33E3ED953FACB78606B08CBFDFE6DAF53E9B5BC3C3E85F95C334F38C65",

"uniqueIdAttributes": [

[

"ab1",

"83D33E3ED953FACB78606B08CBFDFE6DAF53E9B5BC3C3E85F95C334F38C65",

"83D33E3ED953FACB78606B08CBFDFE6DAF53E9B5BC3C3E85F95C334F38C65"

],

[

"ab2",

"83D33E3ED953FACB78606B08CBFDFE6DAF53E9B5BC3C3E85F95C334F38C64",

"83D33E3ED953FACB78606B08CBFDFE6DAF53E9B5BC3C3E85F95C334F38C64"

],

[

"ab3",

"83D33E3ED953FACB78606B08CBFDFE6DAF53E9B5BC3C3E85F95C334F38C63",

"83D33E3ED953FACB78606B08CBFDFE6DAF53E9B5BC3C3E85F95C334F38C63"

],

[

"ab4",

"83D33E3ED953FACB78606B08CBFDFE6DAF53E9B5BC3C3E85F95C334F38C62",

"83D33E3ED953FACB78606B08CBFDFE6DAF53E9B5BC3C3E85F95C334F38C62"

],

[

"ab5",

"83D33E3ED953FACB78606B08CBFDFE6DAF53E9B5BC3C3E85F95C334F38C61",

"83D33E3ED953FACB78606B08CBFDFE6DAF53E9B5BC3C3E85F95C334F38C61"

]

],

"ownershipId": "83D31E3ED952FACB78606B08CBFDFE6DAF53E9B5BC3C3E85F95C399B28C66",

"ownerIdList": [

"4A56E33E9D718571CED220A7347B96FE43DF4E51",

"A7576C8A328EEE4BF69589DDB71099250316FF19"

],

"controlId": "83D31E3ED952FACB78606B08CBFDFE6DAF53E9B5BC3C3E85F95C314F99B28C66",

"controlIdList": [

"4A56E33E9D718571CED220A7347B96FE43DF4E51",

"A7576C8A328EEE4BF69589DDB71099250316FF19"

],

"ownershipTokenId": "83D31E3ED952FACB78606B08CBFDFE6DAF53E9B5BC3C3E85F95C314F99B28C66",

"ownershipTokenAttributes": [

"83D31E3ED952FACB78606B08CBFDFE6DAF53E9B5BC3C3E85F95C314F99B28C65",

"83D31E3ED952FACB78606B08CBFDFE6DAF53E9B5BC3C3E85F95C314F99B28C61"

],

"ownershipTokenQuantity": 3,

"controlTokenId": "83D31E3ED952FACB78606B08CBFDFE6DAF53E9B5BC3C3E85F95C314F99B28C66",

"controlTokenAttributes": [

"83D31E3ED952FACB78606B08CBFDFE6DAF53E9B5BC3C3E85F95C314F99B28C65",

"83D31E3ED952FACB78606B08CBFDFE6DAF53E9B5BC3C3E85F95C314F99B28C61"

],

"controlTokenQuantity": 5,

"identityRecoveryIdList": [

"4A56E33E9D718571CED220A7347B96FE43DF4E51",

"A7576C8A328EEE4BF69589DDB71099250316FF19"

],

"recoveryCondition": 2,

"yesVotesRequiredToPass": 2

}

This way, you do not have to have to enter all this data each time you do a curl request. Now, in the same folder, run both the gatekeeper and ballot javascripts, as follows:

**node ballot2.js**, and **node gatekeeper\_v2.js**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Step 3: *Send the post request for “/gatekeeper”*

Recall in Step 2 we put the data in the app to make testing easier. So your post request need not contain coid data. Run this command:

curl --request POST 'http://10.100.99.207:3000/gatekeeper'

**RESULT** I put in a console.log statement “you reached the /gatekeeper endpoint” which showed up. So we know that this endpoint is successfully reached.

Now, this endpoint listens for events and logs when they are reached. Since the scope of this function is to send data to COID, and event listening can be done separately, I will leave it commented out. For even if we kept event listening, the connection would be pre-emptively cut off with the message “empty reply from server”, so it wouldn’t work.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Step 4: *Send the get request*

curl --request GET 'http://10.100.99.207:3000/'

Verified that the endpoint is reached, but it doesn’t work. What does

res.render("index", { status: obj.data });

this do? obj is unavailable.

Step 5: *Send the get request*

Gatekeeper endpoint doesn’t post validators

curl --request GET 'http://localhost:3000/'