Shareholder Ballot

Shareholder Ballot is a decentralized application (dApp) meant to emulate the real-world practice of shareholders voting in company elections in matters that directly affect their stock ownership. Some companies grant shareholders one vote per share owned, while other companies grant shareholders only one vote. These voting rights allow shareholders to influence the company's corporate direction. (More information about shareholder voting can be found here.)

Quickstart

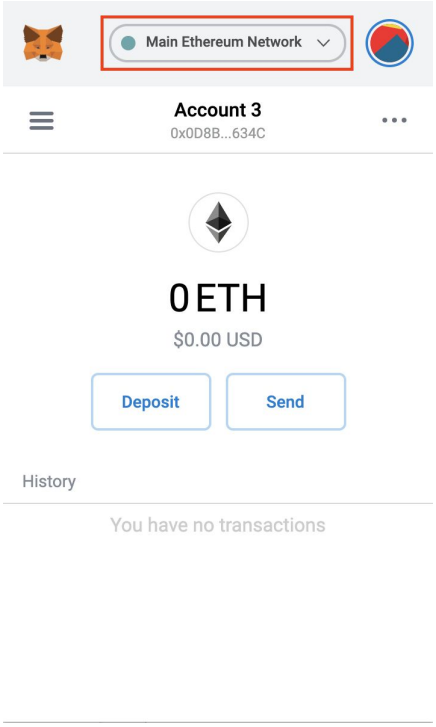
Step 0: Prerequisites

Make sure you have the following prior to attempting to run the dapp:

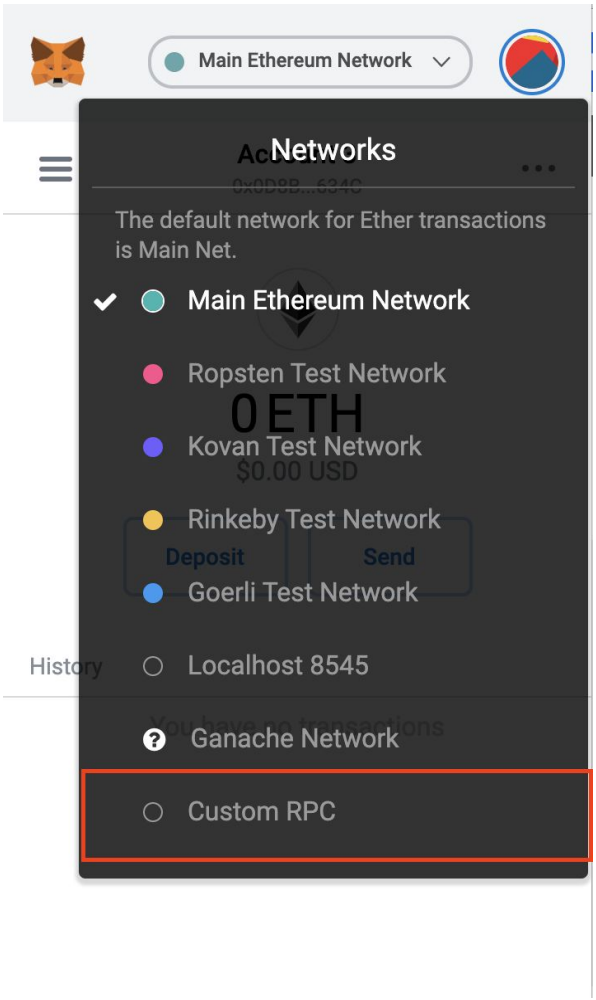
- Truffle
- Ganache
- MetaMask Google Chrome Extension
- Node.js & npm

Hook Up MetaMask to Ganache Network

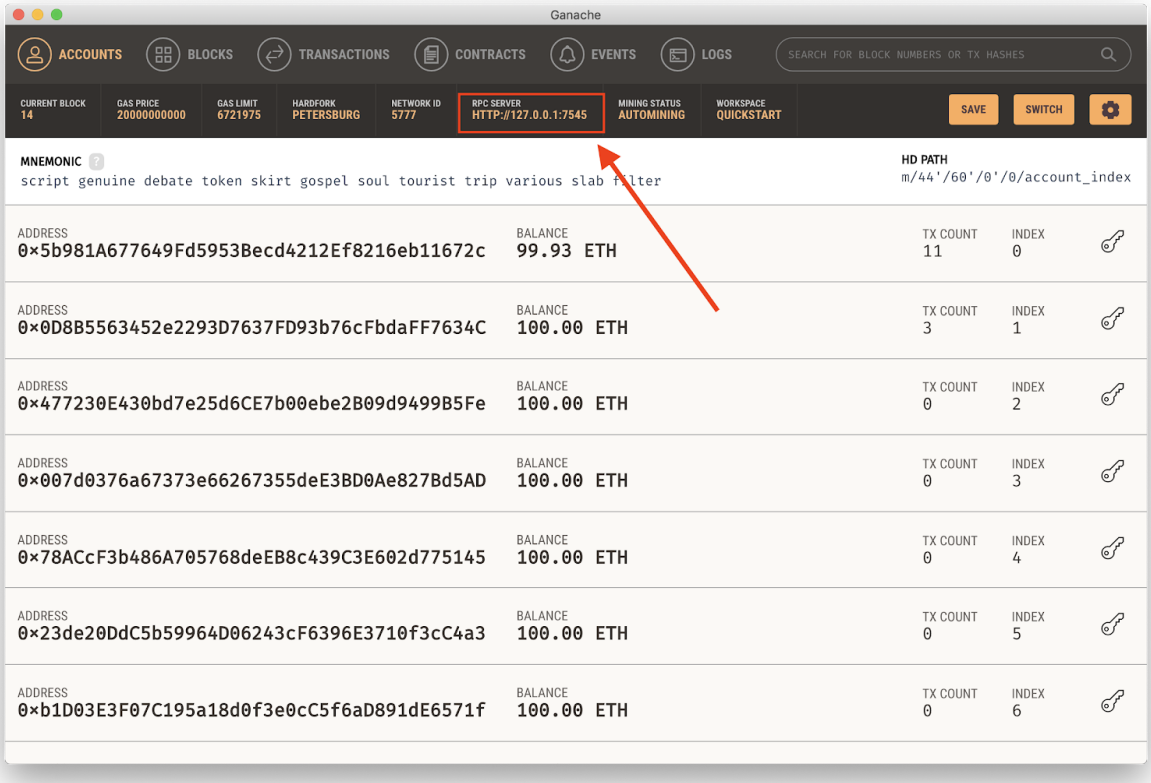
If you already do not have a network that points to your Ganache network, you must set one up. Open up your MetaMask and click on your network name.

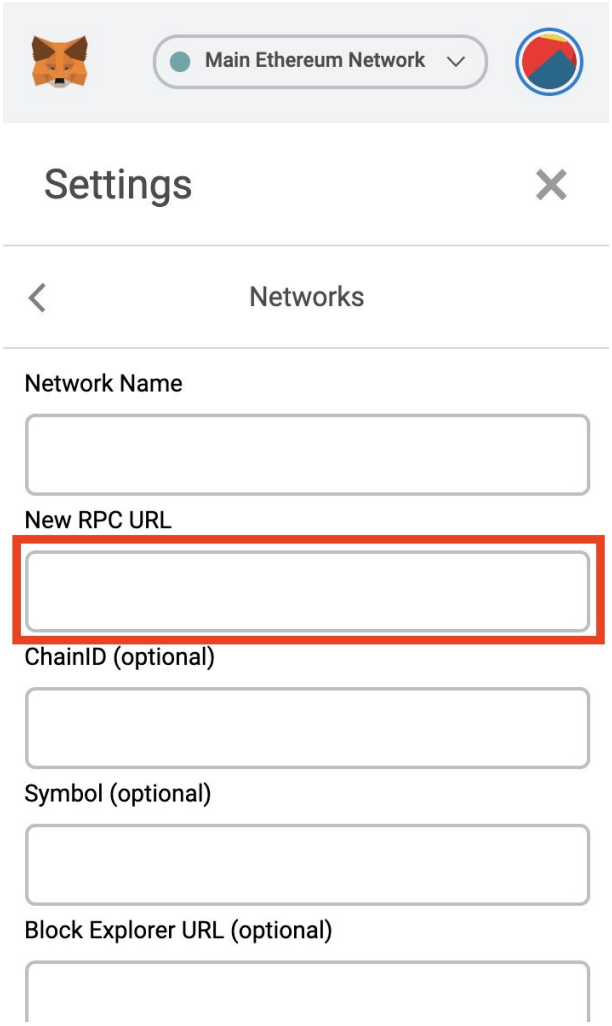


Then click on **Custom RPC**:



Then go to **Ganache**, check the **RPC Server**, then take this value and put it in the **New RPC Value** field within MetaMask. Name your new network whatever you like:

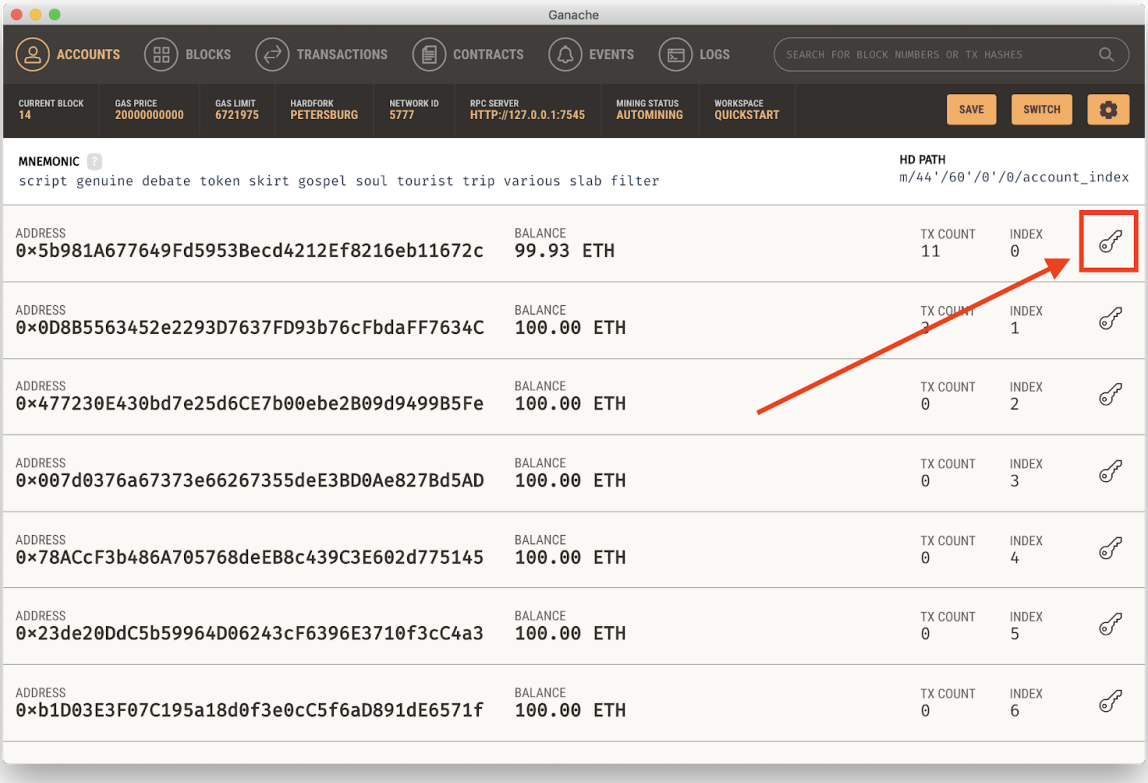


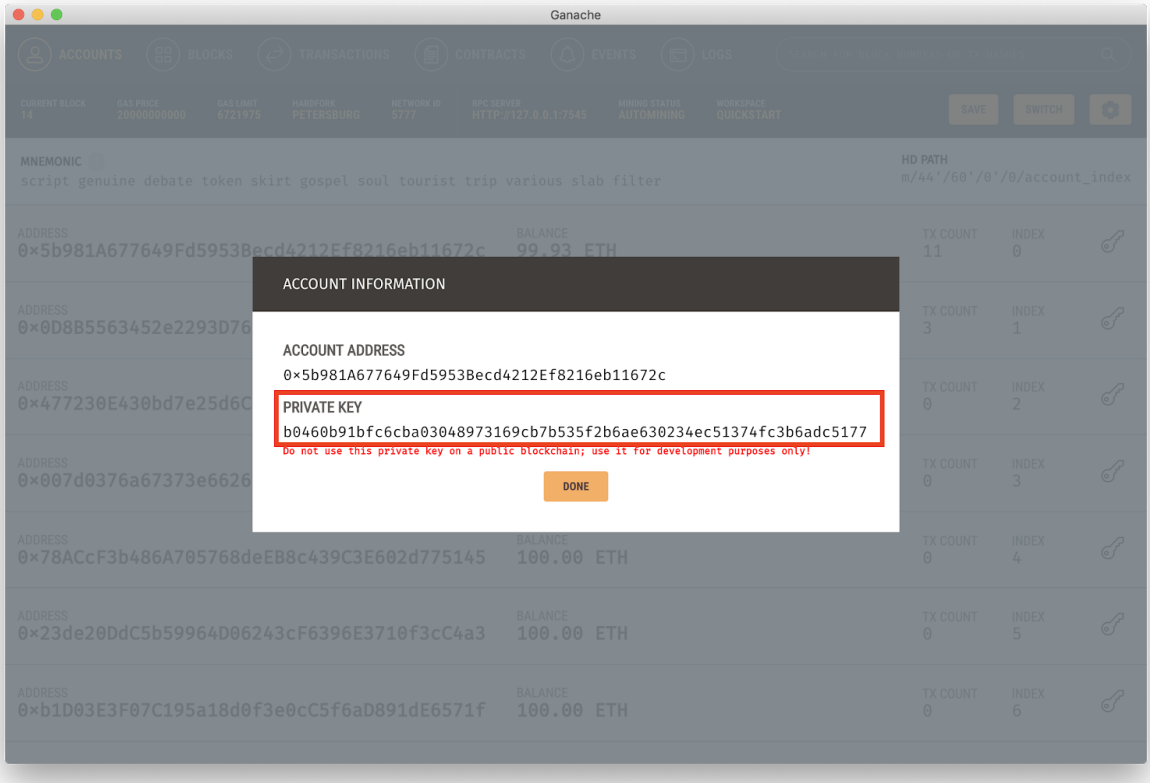


Then click **Save** at the bottom to create this new network.

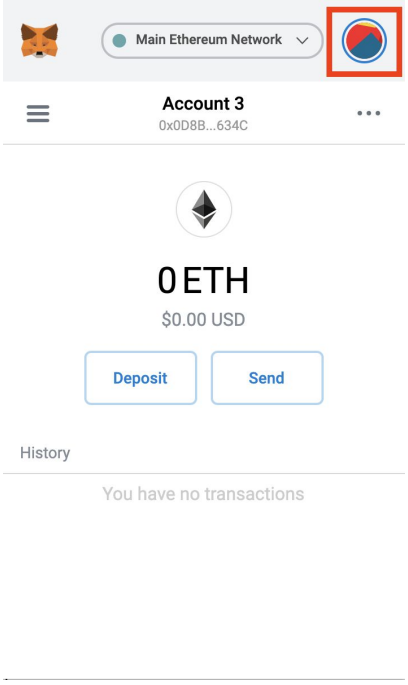
Create Accounts in MetaMask

The next step is to create accounts in MetaMask that are linked to the accounts in Ganache. Do this several times for several accounts (3 recommended). You're going to need the **Private Keys** from the accounts. To get the private keys, navigate to Ganache and click the **key icon** next to the addresses to get the private key.

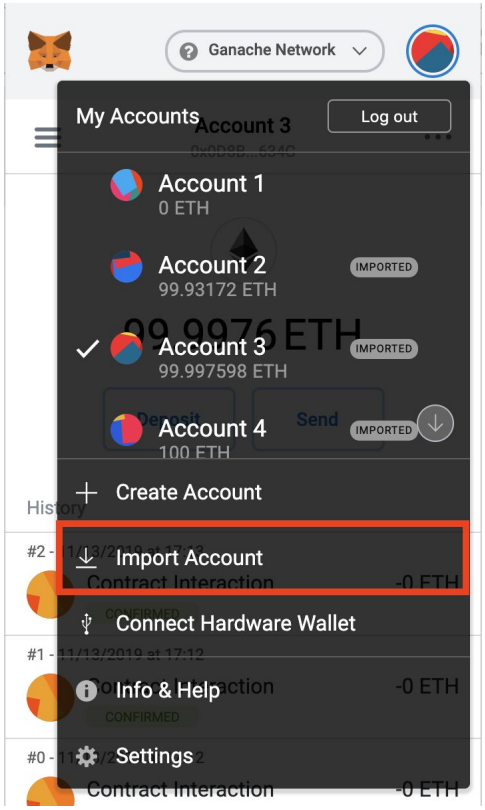

Ganache
ACCOUNTS
BLOCKS
TRANSACTIONS
CONTRACTS
EVENTS
LOGS
SEARCH FOR BLOCK NUMBERS OR TX HASHES
CURRENT BLOCK 14
GAS PRICE 20000000000
GAS LIMIT 6721975
HARDFORK PETERSBURG
NETWORK ID 5777
RPC SERVER HTTP://127.0.0.1:7545
MINING STATUS AUTOMINING
WORKSPACE QUICKSTART
SAVE
SWITCH
MNEMONIC
script genuine debate token skirt gospel soul tourist trip various slab filter
HD PATH
m/44'/60'/0'/0/account_index
ADDRESS 0x5b981A677649Fd5953Becd4212Ef8216eb11672c
BALANCE 99.93 ETH
TX COUNT 11
INDEX 0
ADDRESS 0x0D8B5563452e2293D7637FD93b76cFbdaFF7634C
BALANCE 100.00 ETH
INDEX 1
ADDRESS 0x477230E430bd7e25d6CE7b00ebe2B09d9499B5Fe
BALANCE 100.00 ETH
TX COUNT 0
INDEX 2
ADDRESS 0x007d0376a67373e66267355deE3BD0Ae827Bd5AD
BALANCE 100.00 ETH
TX COUNT 0
INDEX 3
ADDRESS 0x78ACcF3b486A705768deEB8c439C3E602d775145
BALANCE 100.00 ETH
TX COUNT 0
INDEX 4
ADDRESS 0x23de20DdC5b59964D06243cF6396E3710f3cC4a3
BALANCE 100.00 ETH
TX COUNT 0
INDEX 5
ADDRESS 0xb1D03E3F07C195a18d0f3e0cC5f6aD891dE6571f
BALANCE 100.00 ETH
TX COUNT 0
INDEX 6



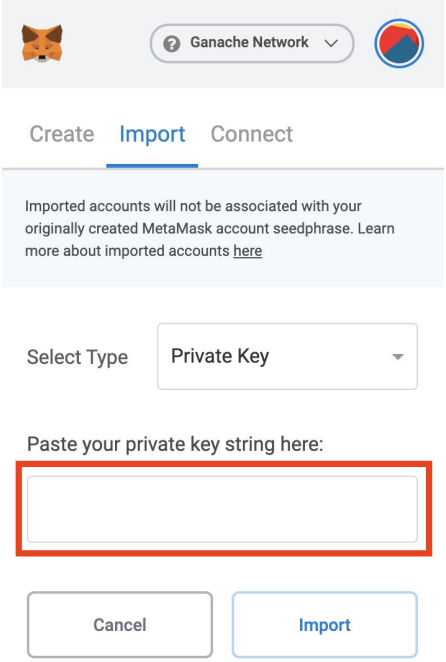
Copy the private key, then navigate over to your browser's MetaMask extension and then click on the account icon in the top-right.



Click on **Import Account**



Then paste the private key you had from Ganache.



Once you hit **Import**, you should have an account! Repeat this several more times as many times as you want for more accounts.

NOTE: You must have at least an account hooked up to the address in Ganache index 0 – this address will act as the “chairperson” for the ballot.

Step 1: Launch Ganache

Launch Ganache and hit **Quickstart**. If Ganache was already open, go to **Settings** and hit **Restart**. If you hit **Restart** on Ganache, you must click **Reset Account** in MetaMask as well for any accounts you imported from Ganache: **Account Icon > Settings > Advanced > Reset Account**.

Step 2: Compile Smart contract

Within the project root, there are two directories: **ballot-app** (client-side) and **ballot-contract** (smart contract). Switch into **ballot-contract**:

```
$ cd ballot-contract/
```

Then use Truffle to compile the smart contracts. The resulting compiled smart contracts will be stored in a new directory: **ballot-contract/build/contracts/**

```
$ truffle compile
```

Step 3: Deploy Smart Contract

Now that the smart contracts have been compiled, it is time to deploy them to Ganache. Run the following command to deploy the smart contract to Ganache:

```
$ truffle migrate --reset
```

Check Ganache to make sure that it actually deployed. If the address at index 0 has an Ether balance less than 100, this means that it was deployed to Ganache.

Step 4: Copy the Compiled Contracts to ballot-app

Since our front end depends on the compiled contracts, we actually need to get them copied over to **ballot-app:**

```
$ cp -a build/contracts ../ballot-app/src/
```

Step 5: Install Node Modules

Now that we have deployed our smart contract, we need to install the dependencies for our client side. First thing we need is to change our directory over to **ballot-app**:

```
$ cd ../ballot-app
```

Now we need to install our node modules:

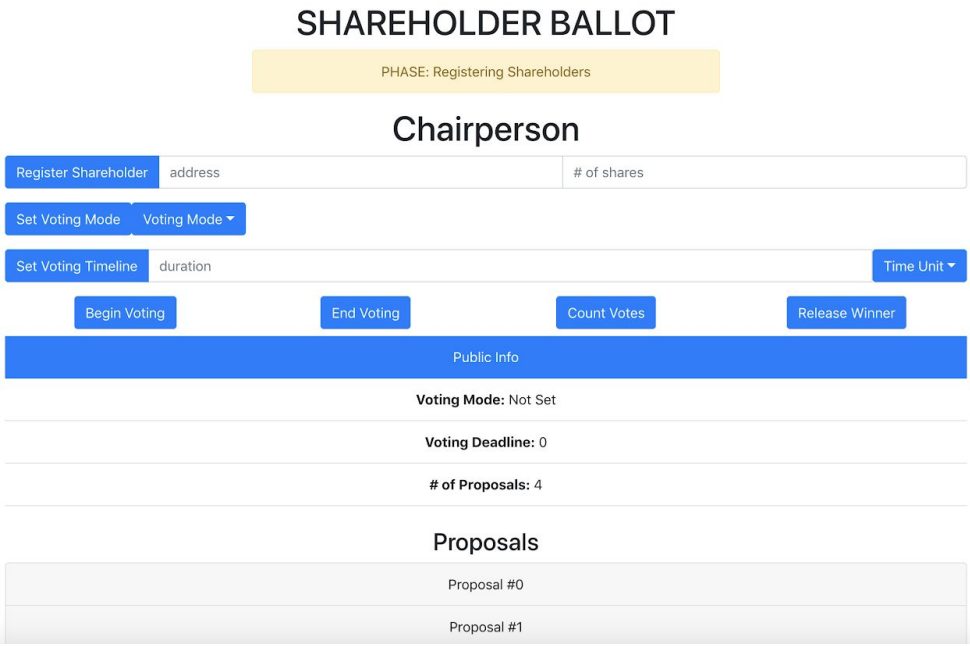
```
$ npm install
```

Step 6: Run Client-Side Front End

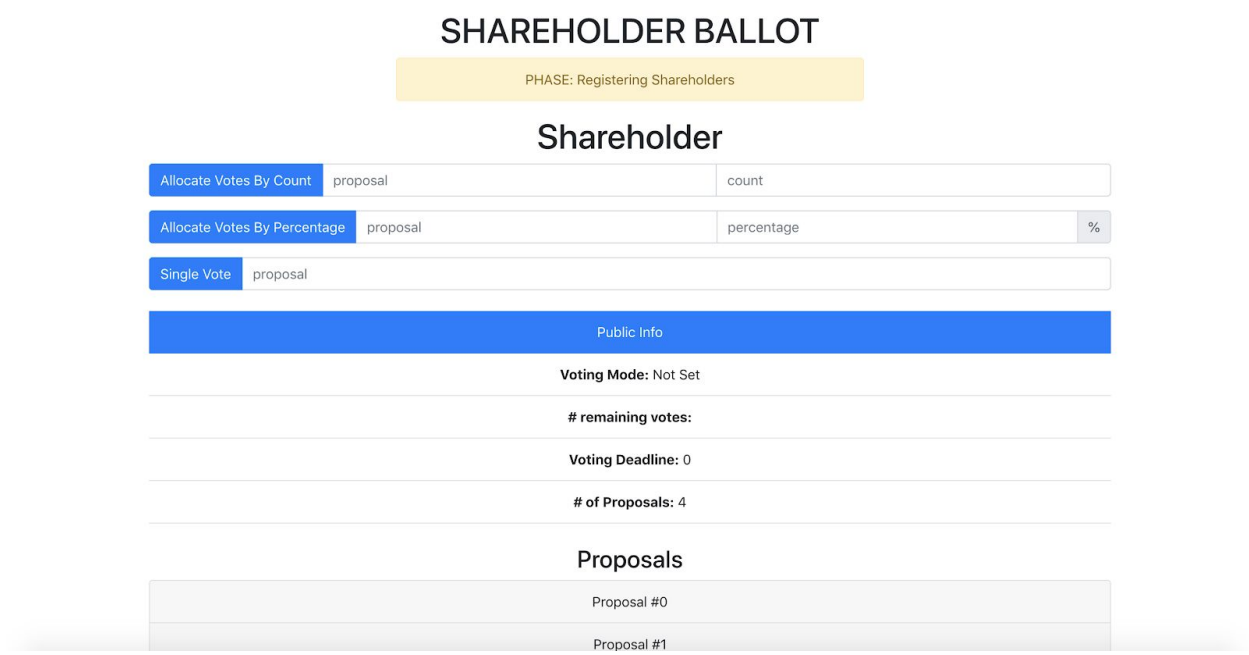
Now that the node modules have been installed, all we need to do is run our front-end:

```
$ npm start
```

If, on MetaMask, your account is set to the deployer of the Smart Contract (index 0 – 0x5b98…), you are the **chairperson** – so your frontend will look like this:



Otherwise, you are a **shareholder**, so your frontend will look like this:



Note: When switching between accounts, hit refresh to see the changes in the user interface.

Step 7: Play Around with Application

Feel free to play around with the application! Using MetaMask to switch between **Chairperson** and **Shareholders**.

As a chairperson, you can:

- Register shareholders (by address and number of shares)
- Set voting mode
    - Each shareholder gets one vote per share
    - Each shareholder gets only one vote
- Set the voting timeline and deadline
- Change the phase of voting:

As a shareholder, you can:

- Allocate a number of votes to a proposal
- Allocate a percentage of votes to a proposal
- Vote on a single proposal

Running Smart Contract Unit Tests

Unit tests for testing the business logic of the smart contract can be found under **ballot-contract/test/ShBallot.js**. To run tests, make sure that the smart contract is deployed to Ganache:

```
$ truffle migrate --reset
```

Then run the following command:

```
$ truffle test
```

And the unit tests should pop up:

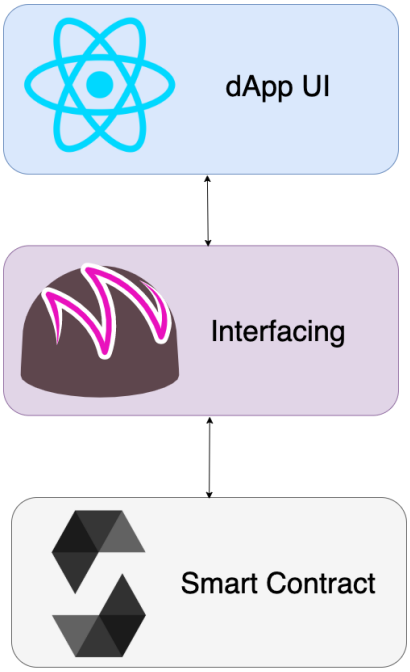
```
Compiling your contracts...
===========================
> Everything is up to date, there is nothing to compile.



  Contract: ShBallot
    ✓ Should be in Phase.Regs (1) upon deploying the smart contract (65ms)
    ✓ Should be able to have chairperson register other accounts and they can access the correct number of votes (224ms)
    ✓ Should not let a voter begin voting -- only chairperson can begin voting (500ms)
    ✓ Should not let a non-registered voter tries to access their number of votes (47ms)
    ✓ Should be in Phase.Vote (2) (431ms)
    ✓ Should have voters have only one vote if in VoteMode.OneVote (1) (298ms)
    ✓ Should not let a voter allocate votes by number if in VoteMode.OneVote (543ms)
    ✓ Should not let a voter allocate votes by percentage if in VoteMode.OneVote (498ms)
    ✓ Should have the correct number of votes after calling allocateVotesByPercentage (955ms)
    ✓ Should not let a voter see the winner early (724ms)
    ✓ Should have the correct proposal as winner (784ms)
    ✓ Should not let voters vote after the deadline has passed (3343ms)
    ✓ Should not let voters vote if not in Phase.Vote (472ms)
    ✓ Should not let voters allocate more votes than they have (523ms)
    ✓ Should not let voters vote more than once for VoteMode.OneVote (841ms)
    ✓ Should not let chairperson start voting without anyone registered (266ms)
    ✓ Should not let chairperson start voting without setting voting timeline (306ms)
    ✓ Should not let chairperson start voting without setting voting mode (289ms)


  18 passing (15s)
```

Tech Stack



- Client-side/front-end
  - React – JavaScript library for developing user interfaces
  - Drizzle – front-end development tool, part of the Truffle suite
- Smart Contract
  - Solidity – object-oriented programming language for writing smart contracts
- Development
  - Ganache – used to fire up personal Ethereum blockchains to help smart contract testing and development
  - MetaMask – used to run Ethereum dApps within a web browser

The case for using React and Drizzle

As you have already noticed, this application uses a tech stack different than that in the example **Ballot.zip** provided to the CSE 426 class. The Ballot provided to us runs using Node.js, Express, and an index.html.

We decided to work with React and Drizzle for the following reasons.

Modularity and Ease of Development

React allows us to create reusable components which get rendered as HTML once we run the application. It makes it much easier to create front-end user interfaces that have functionality

and state associated with it. And as we are creating individual components, this means more reusability.

Drizzle is a library we found while researching how to hook up our React front-end to the deployed smart contract. We found that Drizzle would accomplish exactly everything we needed for this dApp: synchronizing contract data and transaction data. Drizzle will handle most of the heavy lifting – it acts as a middle layer between our front-end and our smart contract. It is merely an extension of Web 3.0's contracts.

React Community

The React community is large and only growing. React is based off of Node.js, which means that we can use Node.js modules within React rather than just use Express. This is extremely helpful and it keeps our code uniform.

New and Popular Technology/Tech Stack

React is rising in popularity and is high in demand. The demand for React engineers or software engineers that know React is growing, so we wanted to create an application that reflected the future of front-end user interfaces.