Technical Specification

CA326 3rd Year Project

- **Project Title:** Decentralized Voting App
- **Student 1 Name:** Theo Coyne Morgan **ID Number:** 17338811
- **Student 2 Name:** Ciarán Palmer **ID Number:** 17425304
- **Project Supervisor:** Annalina Caputo

Table of Contents

1. Introduction

1.1 Overview

This technical specification is an updated complete description of the voting app system, its modules and 3rd party components. This document outlines the architecture, high level design and development issues of the project.

‘Devote’ is a decentralized voting app that utilizes the Ethereum blockchain, ReactJS and a database to create an anonymous voting system aimed at university societies. It allows the creation of new societal elections, adding candidates, voting in elections, management of society members and automatic calculation of votes.

The system uses the blockchain for its many advantages to voting, for example, information stored on it is immutable and cryptographically secured by hundreds of nodes on the blockchain. This prevents tampering of data or falsified votes.

1.2 Glossary

Blockchain

A blockchain is a growing list of records, called blocks, that are linked using cryptography. Each block contains a cryptographic hash of the previous block, a hash of the current block, and the transaction data (in our case, a vote) of the block.

Smart contract

A smart contract is a piece of code that is stored on the blockchain. This piece of code is then fetched and executed when required. As it is stored on the blockchain it is distributed and immutable.

Solidity

Solidity is a statically-typed, object-oriented programming language developed by Ethereum for writing smart contracts. It is designed to have a similar syntax to

JavaScript. It is used for implementing smart contracts on various blockchain platforms, most notably, Ethereum.

Ethereum

Ethereum is an open-source, public, blockchain-based distributed computing platform and operating system featuring smart contract functionality. Ethereum provides a decentralized virtual machine, the Ethereum Virtual Machine (EVM), which can execute smart contracts using an international network of public nodes.

Ganache

Ganache is 3rd party software that creates a private Ethereum blockchain to run tests, execute commands, and inspect state while controlling how the chain operates. It provides the ability to perform all actions you would on the main chain without having to use real cryptocurrency. It provides convenient tools such as advanced mining controls and a built-in block explorer.

DApp

A decentralized-app (DApp) is a computer application that runs on a distributed system, i.e. the blockchain.

Metamask

Metamask is a browser extension that allows the user to run Ethereum Dapps in your browser without running a full Ethereum node. MetaMask includes a secure identity vault, providing a user interface to manage your identities on different sites and sign blockchain transactions.

Web3 JS (Ethereum JavaScript API)

Web3 JS is a collection of libraries which interact with a local or remote Ethereum node, using an HTTP or IPC connection.

NoSQL

A NoSQL database provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases.

2. System Architecture

This section describes the high level overview of the entire voting system including its modules and 3rd party components. There are three main components that function in unison to create the decentralized voting app:

2.1 Blockchain

This component consists of an Ethereum blockchain network, an Ethereum node and a Smart contract deployed on the blockchain network. The Ethereum node is a client on the network that communicates with the smart contract. The smart contract is written in Solidity - a programming language designed specifically for writing smart contracts on the Ethereum blockchain. The contract contains logic for adding new elections and candidates, casting votes and storing voting information on the blockchain. Ganache was used in this project for development purposes. Ganache provides a personal Ethereum blockchain network to deploy contracts and execute commands. It provides the ability to perform all actions you would on the main chain without the cost. Metamask is an Ethereum wallet that runs in your browser and allows you to interact with Ethereum-enabled web apps such as Devote.

2.2 ReactJS

React is a JavaScript library for building user interfaces and building web apps. In the voting system it serves as the front-end user interface and handles requests to both the database and to the blockchain contract.

Web3.js is used as a Javascript library to interact with a local or remote ethereum node. It is used as an API to the blockchain contract. It works in conjunction with React to make calls to the contract functions and send or receive data.

2.3 Database

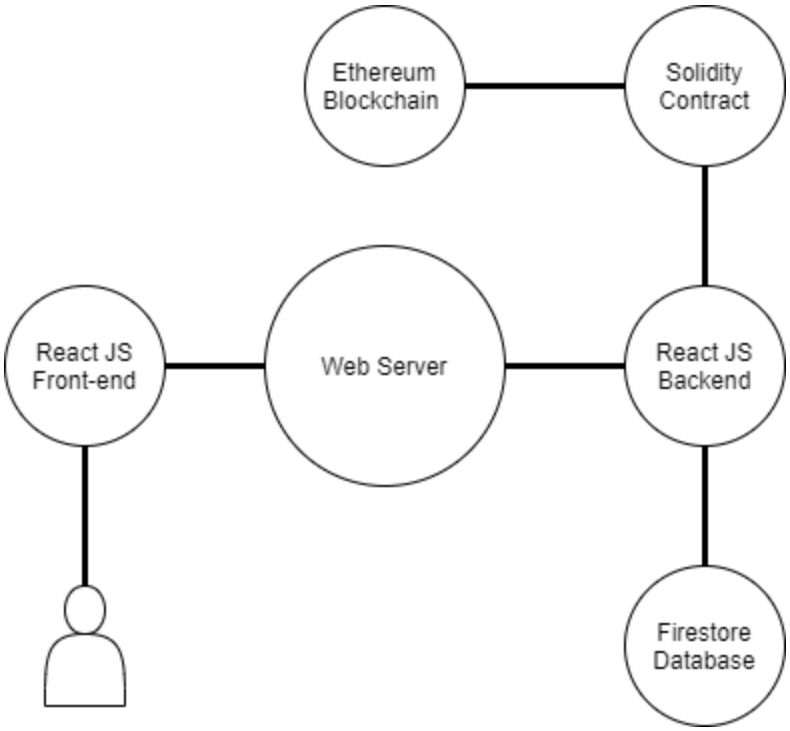
Firebase is a mobile and web application development platform provided by Google. Firestore is a database provided by Firebase. It is a fast, NoSQL document database that is used to store various information such as invite codes, election expiry dates,

users and societies. It acts as an extra layer of control above the blockchain and allows for fast data retrieval and storage.

3. High-Level Design

High level System overview

React components create the user interface and manage requests to and from the backend. They are located in the directory `code/src/components`.

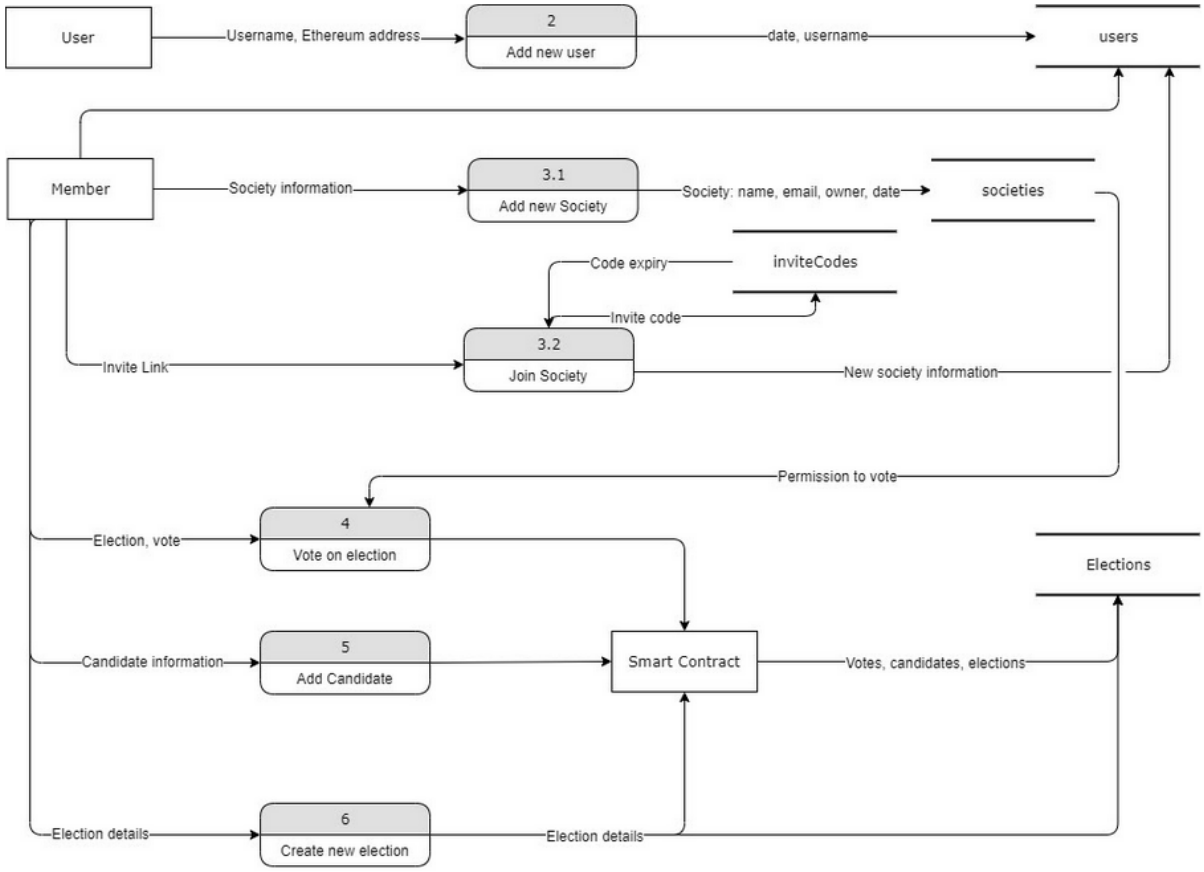


Data flow Diagram

This diagram illustrates how data is sent and received in the system. Data stores such as `user`, `societies` and `inviteCodes` are located on the Firestore Database. The Smart Contract component interacts with the blockchain and stores information such as candidates and election details.

The contract is written in Solidity and is located in the directory `code/src/contracts/VotingApp.sol`

React is used as the client user interface. It allows users to easily execute functions and interact with the blockchain.



4. Problems and Resolution

This section includes a description of any major problems encountered during the design and implementation of the system and the actions that were taken to resolve them.

4.1 Loading Times

At the beginning of our project, our system was so simple that everything loaded almost instantaneously. As we added more and more content, it started taking a few

seconds to retrieve data from both the blockchain and the database. This resulted in a blank page until the data finally loaded. The solution to this problem was to add a new loading state for every major function in the system. When a component is being loaded, the loading state is set to true. When the component is finished loading, the loading state is set to false. All components on the site are conditionally rendered based on the value of the loading state.

A custom loading animation is played whilst a component is being loaded to give the system a very app-like feel.

4.2 Data Storage Problem

We couldn't store all the information needed on the blockchain because the more information on the blockchain, the longer it takes to query it. We needed a database to implement all the required functionality. The most optimal database to use in our case was the Firestore hosted by Google. This gave us a free website to host our application and a database.

The Firestore has its own unique syntax for retrieving information which we both have never used. We had to go through the documentation on the Firestore website to learn how to query and add data to the database. It took roughly two days to overcome this learning curve and competently write data to and from the Firestore.

4.3 Storing Local Data

In the beginning I used a naive solution by making a call to the database every time I needed some information. This slowed the program down significantly and made the pages feel very sluggish as they took so long to load. A better solution to this problem is to store all the information required from one database call in a React State. The state doesn't get reset unless the page is reloaded. The information from this state can be passed down to lower level components using React Props. When a page is loaded, the system makes all the major database calls and stores the information in a number of different state variables. This data can then be loaded on any page almost instantaneously. The initial load time takes longer because of this, but only about three seconds. This is a very good trade off as we removed most of the loading required after the site is loaded for the first time.

4.4 Start & End Date Input Errors

On the election page, we needed the functionality to set a start and end date for the election. The input for these dates and times had to have detailed error handling to

guide the user through the process. The input in the finished application can detect eight different errors:

- Wrong Start Date Input
- Wrong Start Time Input
- Wrong End Date Input
- Wrong End Time Input
- Start Date must be later than the current date
- Start Time must be later than current time (if date is the same)
- End date must be at least 12 hours after the start date
- Election must have two candidates before it can be started

This was a very challenging aspect to the application as it was very heavy with logic. This component took the most time of all the components in the application. This is because I had to take two dates and two times as inputs. The dates and times has to be constructed into two date objects and compared against one another to ensure there are no errors. The dates had to be stored in the database and the software had to automatically start or end an election based on the current date and time. After nearly a week of working on this component it was finally finished without bugs.

4.5 Changing State whilst Component is Unmounted

This was a problem specific to the React framework that we chose. We both had never used React prior to this project and used a number of naive solutions in the beginning. This later caught up with us as we started receiving errors about changing state whilst the component isn't mounted. To solve this problem, we had to add lots of if/else statements to the start of each component and ensure that a state was mounted before trying to access or change it.

4.6 Asynchronous Functions

We had a lot of issues with data not being retrieved on time due to synchronous calls to the database. We were both new to asynchronous functions and had a slight grasp on the concept. After reading as much information about them as we could, we started implementing them into our database calls to allow the information to be retrieved before processing more of the function. This also solved an important issue where it took two refreshes to change the state of an election. Using asynchronous calls it now changes state in a single refresh.

5. Installation Guide

Installation guide video

You can view the Installation guide video here:

https://www.youtube.com/embed/2tP3DLb9yXk

Requirements

Before proceeding, you will need to have the following packages installed on your machine:

- Npm (https://www.npmjs.com/get-npm)
- Git (https://git-scm.com/)
- Ganache (https://www.trufflesuite.com/ganache)
- Metamask (https://metamask.io/download.html)

5.1 Installing the Application

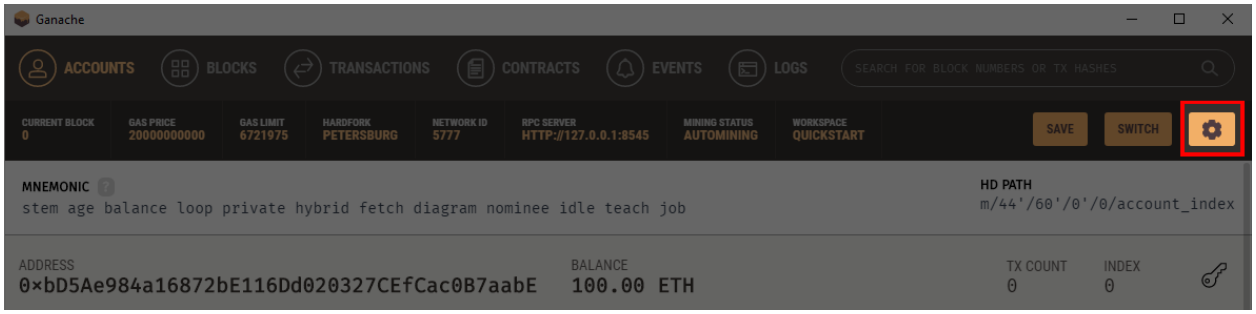
After installing all packages, open command prompt and enter the following commands in a suitable directory:

- `git clone https://gitlab.computing.dcu.ie/coynemt2/2020-ca326-tcoynemorgan-DecentralizedVotingApp.git devote`
- `cd devote/code`
- `npm install --save`
- `truffle migrate --reset`
- `npm run start`

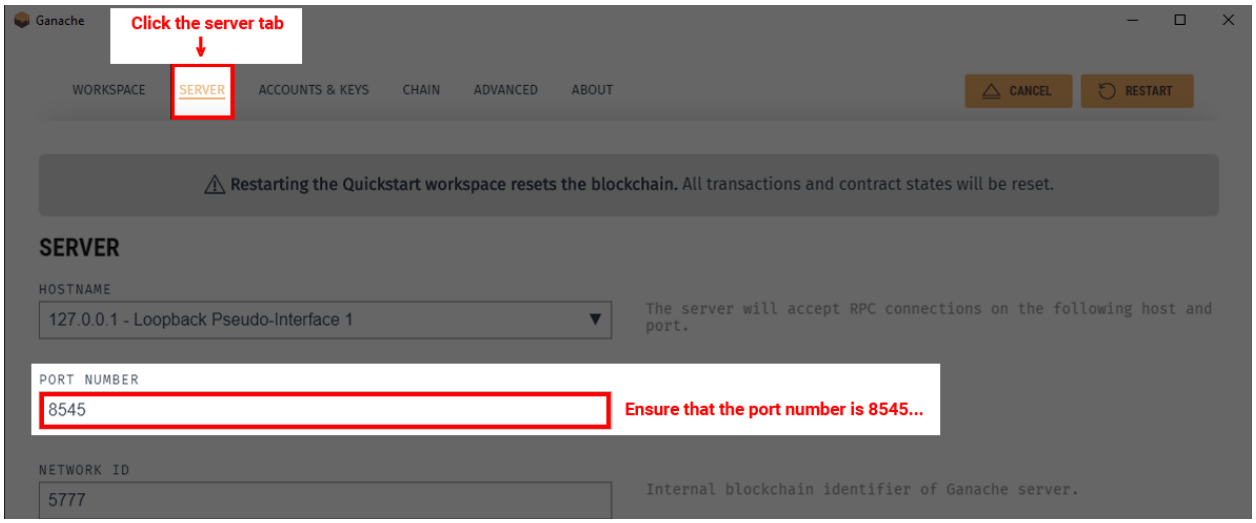
After typing `npm run start` your console will open a webpage on `https://localhost:3000`

5.2 Starting the Blockchain

Once you have **Ganache** downloaded, opening the program will show this screen. **Click Quickstart**. This will bring you to this screen. **Click the Settings Icon** in the **top right** of the window.



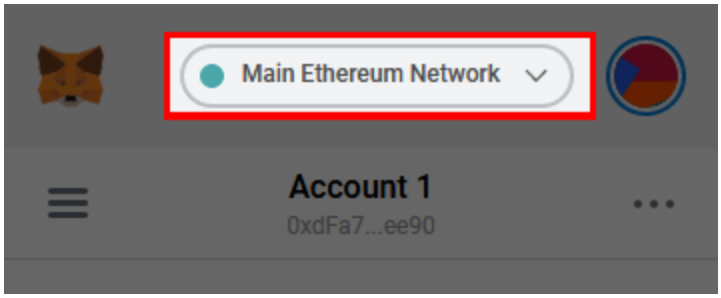
Click Server in the navigation bar at the top and ensure the **Port Number** is equal to **8545**.



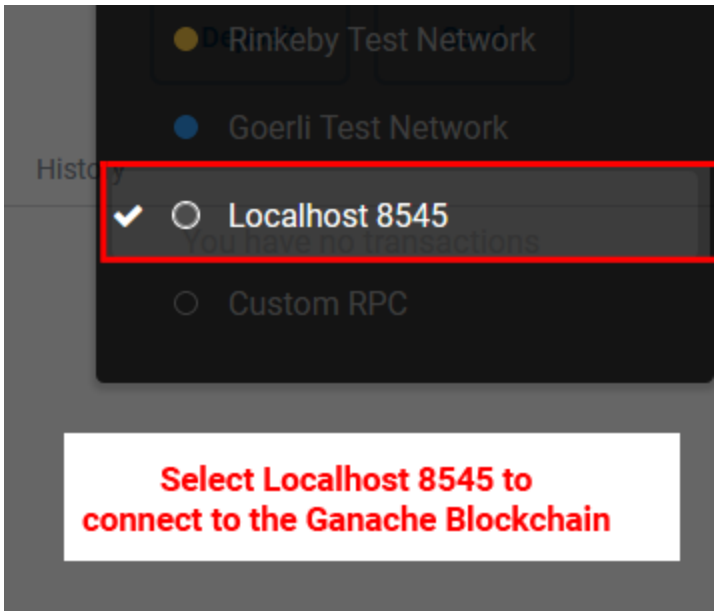
5.3 Connecting to the Blockchain using Metamask

After installing the Metamask browser extension, click ‘Get Started’ once the window opens. Follow the instructions to set up a new account. Once you have finished, click the **small Metamask icon** in the top right corner of your browser.

Select the dropdown menu labeled **Main Ethereum Network** at the top of the Metamask extension.

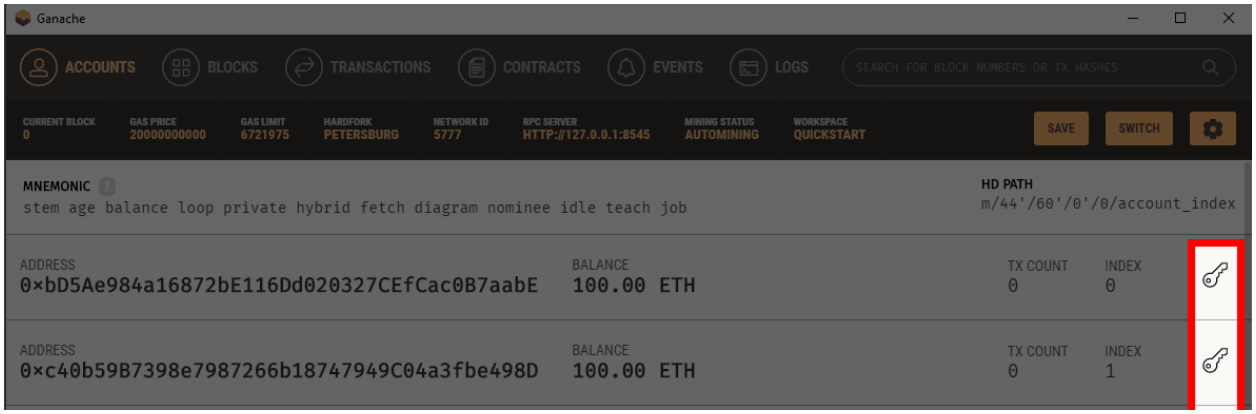


Change the network from **Main Ethereum Network** to **Localhost 8545**

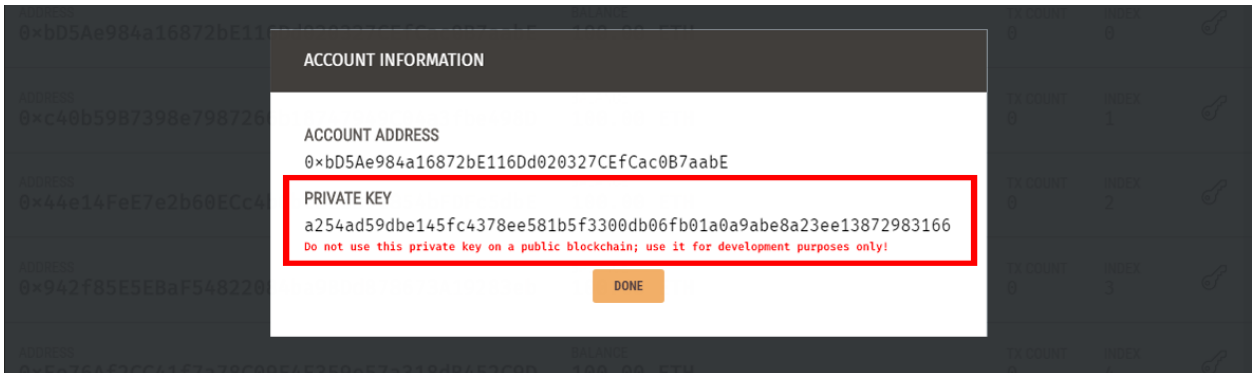


5.4 Importing Accounts from the Blockchain

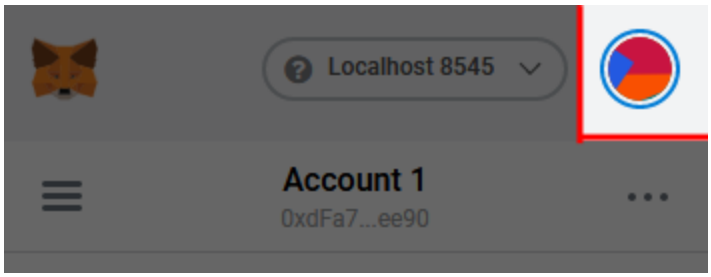
There are ten accounts available with Ganache. To import an account, **click the key icon** next to one of the ten accounts on the **Ganache home screen**.



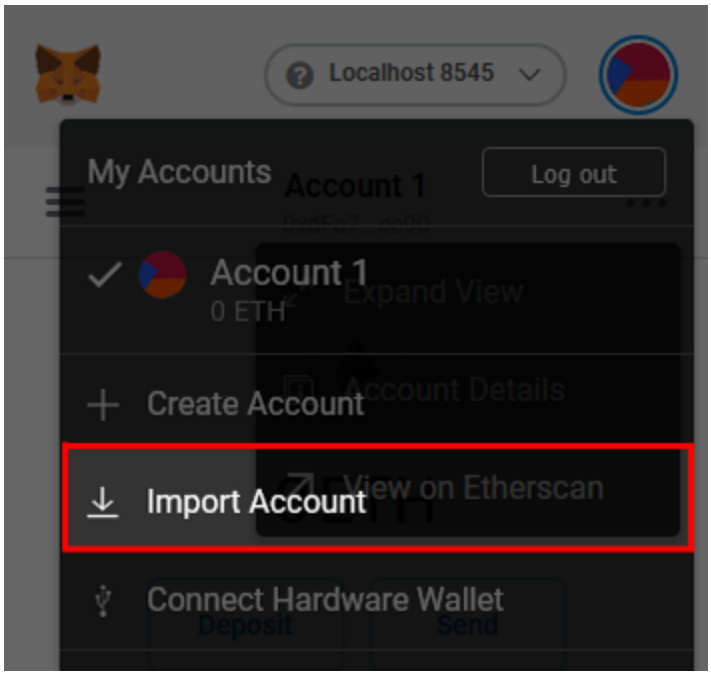
Copy the Private Key from the window that pops up and **click done**.



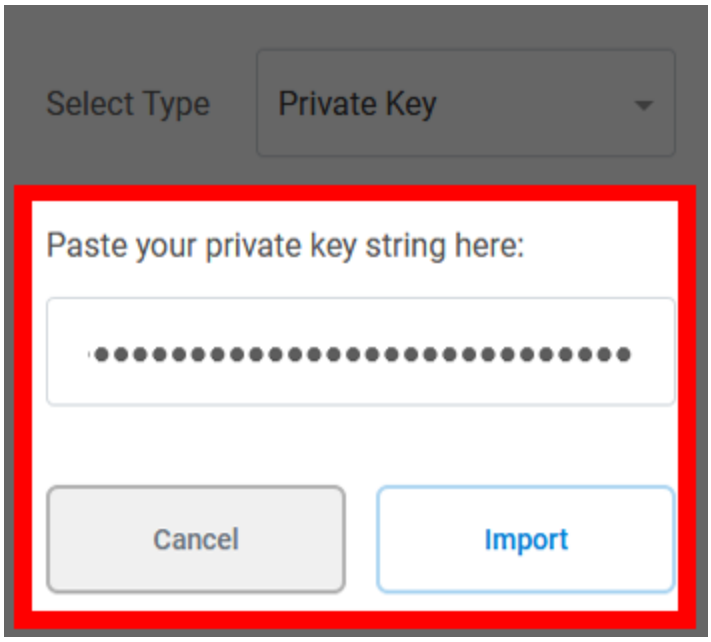
Click the Metamask icon in the top right corner of your browser and **click the circle icon** in the top right of Metamask.



Select the Import Account option from the dropdown.



Paste the Private Key into the highlighted input form. **Click import.**



Multiple accounts can be imported to Metamask from ganache using their unique private keys. You can swap between accounts in the Metamask browser extension and refresh Devote to represent different users.

Once these steps have been completed, you are ready to use Devote.

5.5 Running unit tests

To run unit tests on the Solidity contract first make sure you are in the `code` directory. Then execute the following command:

```
truffle test
```

This executes the unit tests located in the `test` directory. Truffle uses the Mocha testing framework and Chai for assertions on the Solidity smart contract. The tests will run and check that essential system functionality is operational, for example creating elections, adding candidates and voting.