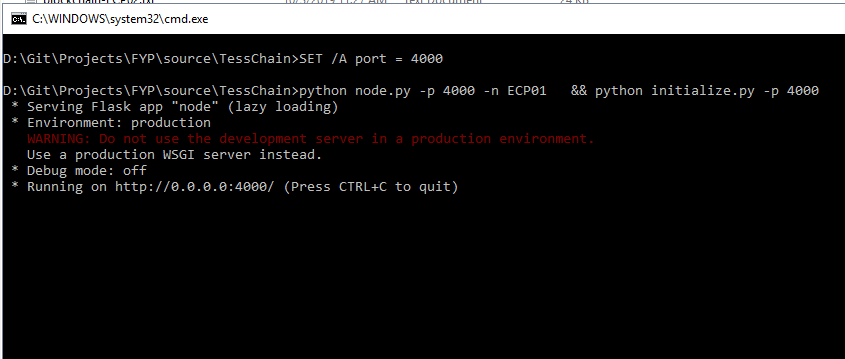
The TessChain API has been developed using “Flask” a popular python micro-framework, it is a third-party Python library for developing APIs. To build the API, many third-party libraries like Flask, Cryptodome, hashlib, JSON and requests have been employed. And the e-voting system is developed on top of the blockchain protocol named DEVOTE. The front end of both mobile and web app is built using Flutter, an open-source application development framework created by Google. It is used to develop applications for Android, iOS, and Web also. And Git is used to version control the whole project.

To start with, first you have to setup the blockchain network. To do such, execute the main.bat file or you can edit it to change the parameters according to your need (port number = 4000 and nodeID = ECP01 default).

Here is an example:

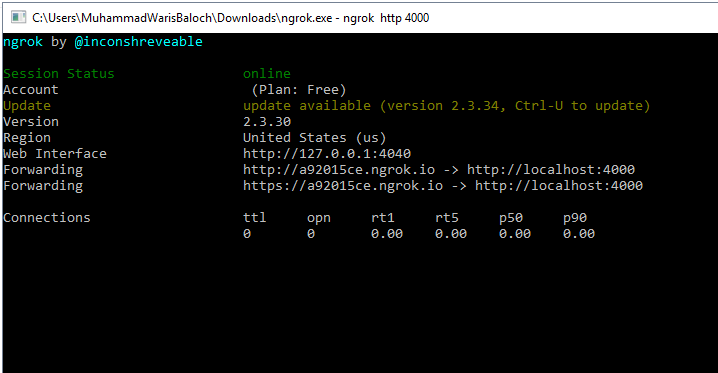


**Implementation on single/multiple nodes:**

If you want to test the API, you can just run multiple nodes on the same PC. And test its endpoints using Postman.

But if you want to implement it over multiple PCs connected over the internet, you can use any tool to create a public HTTPS URL for the API running on a localhost. The software we have used is ngrok for making the local node accessible through the internet ([setting up ngrok](https://ngrok.com/docs))

To expose the server over the internet, the command to used is ngrok http 4000. The port must be the same as the server port.

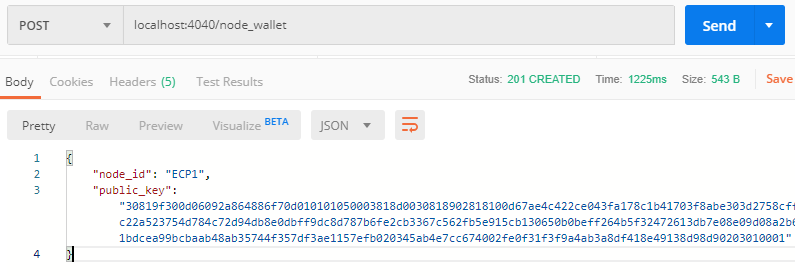


**Testing the endpoints:**

To test the endpoints, again we will be using Postman.

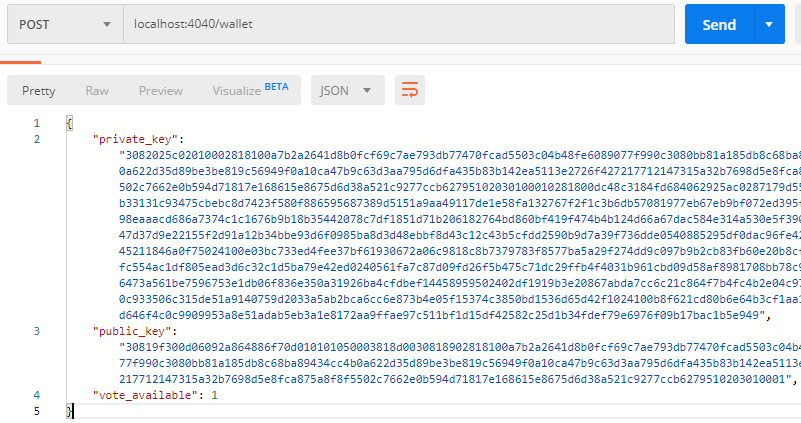
1. **Initializing validation node wallet**

The next step after starting the server will be creating a wallet for the validation node. The endpoint “node\_wallet” is called using POST method to the API for creating the wallet, and it is called automatically when the main.bat file is executed. It responds with the node’s public/private key pair.



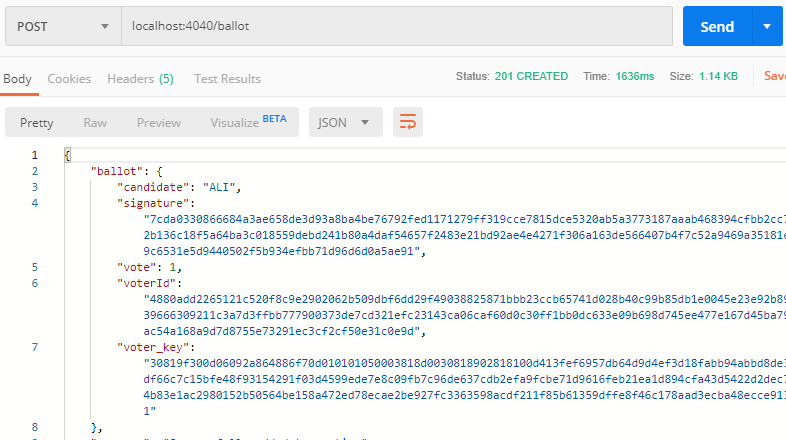
1. **Initializing voter wallet**

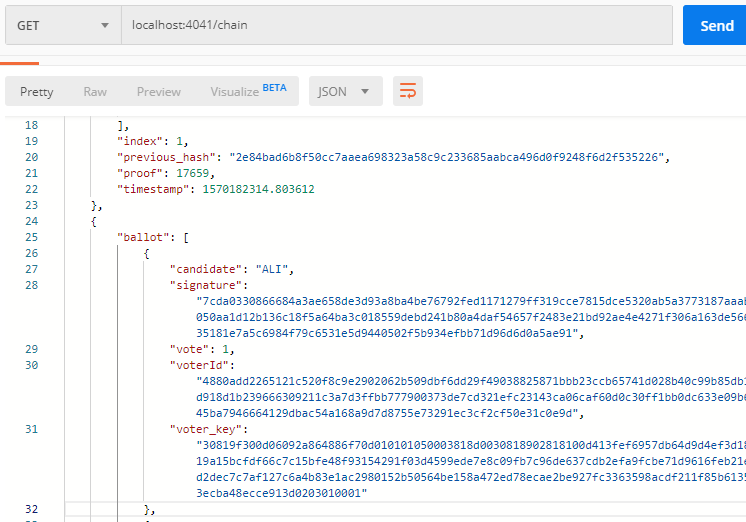
The validation node is setup, and now we move to the voting process. After the voter’s authentication, a wallet has to be created in order to cast a ballot. When the voter logs in the “wallet” endpoint is called and it responds with the voter’s public/private key pair, and the VOTE token place into that wallet by the node.

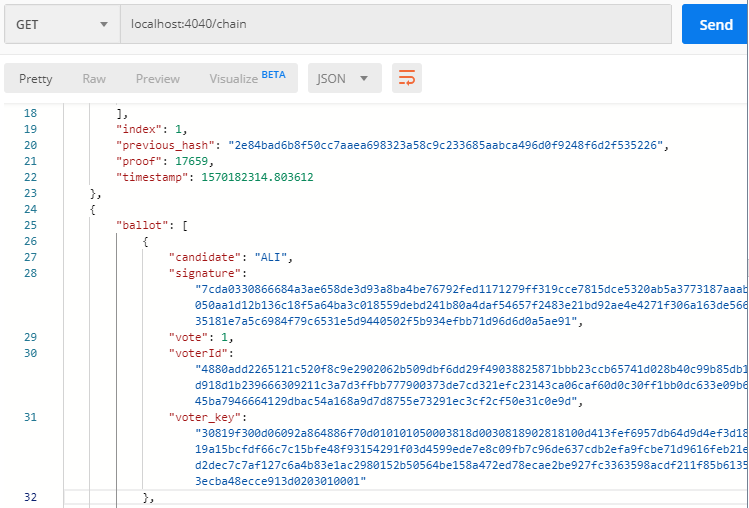


1. **Casting ballot**

After the wallet creation, the voter can select a candidate and cast a ballot. A request is sent to the “ballot” endpoint along with JSON data that contains the voter’s choice, when the voter confirms the ballot and in response the API broadcasts the ballot information (candidate, voter key, signature and encrypted voter ID) to the peer nodes for validation.

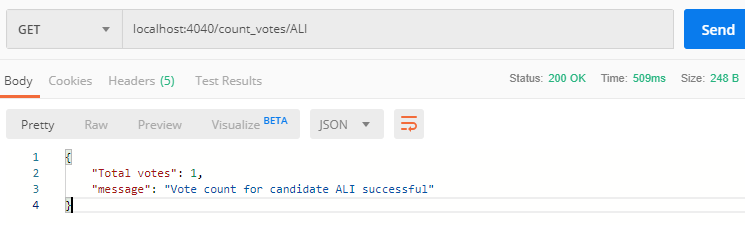






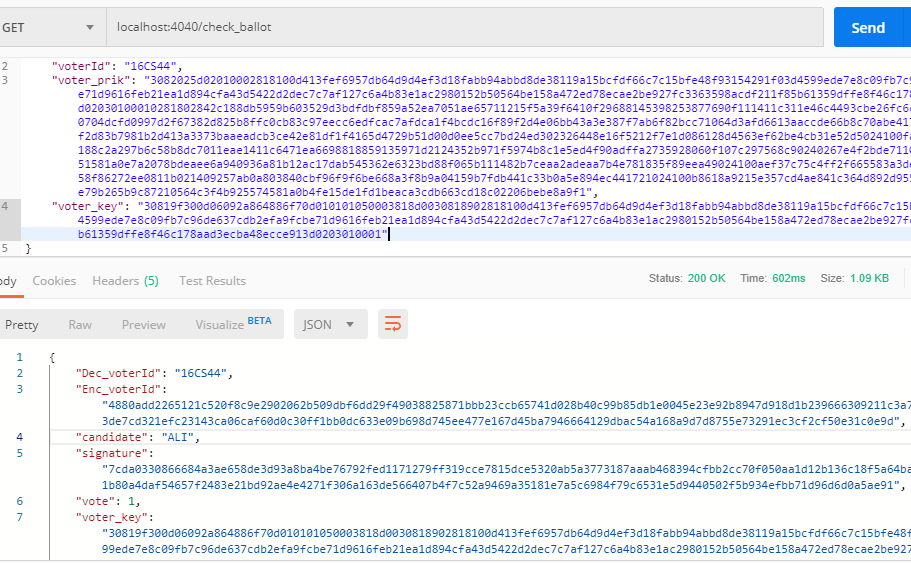
1. **Vote count**

After the voting process ends, the “count\_votes” endpoint will be triggered by the tallying server in order to display the result of the web app. For testing, a GET request is sent to the server in order to count votes for a candidate (ALI in this case).



1. **Ballot verification**

To verify the voter’s ballot, a request is sent to the “check\_ballot” endpoint along with JSON data that contains the voter’s ID, public key and private key. The public key is to improve the searching time, and the private key to decrypt the voter ID and check if the decrypted text matches the ID. Only then it will return the voter’s ballot, show in figure 5.9.



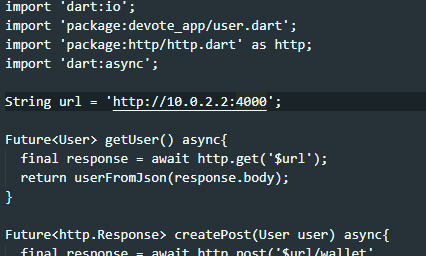
**Configuring the front end:**

The devote\_app folder contains the mobile application source code for Android and iOS both platforms.

Emulator configuration settings:

Name: voting\_tablet\_API\_27  
  
CPU/ABI: Google APIs Intel Atom (x86)  
  
Path: C:\Users\username\.android\avd\voting\_tablet\_API\_27.avd  
  
Target: google\_apis [Google APIs] (API level 27)  
  
Skin: 1280x800  
  
SD Card: 512M  
  
fastboot.chosenSnapshotFile:   
  
runtime.network.speed: full  
  
hw.accelerometer: yes  
  
[hw.device.name](http://hw.device.name" \t "_blank): voting\_tablet  
  
hw.lcd.width: 1280  
  
hw.initialOrientation: landscape  
  
image.androidVersion.api: 27  
  
[tag.id](http://tag.id" \t "_blank): google\_apis  
  
hw.mainKeys: no  
  
hw.camera.front: emulated  
  
avd.ini.displayname: voting\_tablet API 27  
  
hw.gpu.mode: auto  
  
hw.ramSize: 512  
  
PlayStore.enabled: false  
  
fastboot.forceColdBoot: no  
  
hw.cpu.ncore: 2  
  
hw.keyboard: yes  
  
hw.sensors.proximity: yes  
  
hw.dPad: no  
  
hw.lcd.height: 800  
  
vm.heapSize: 48  
  
skin.dynamic: yes  
  
hw.device.manufacturer: User  
  
hw.gps: yes  
  
skin.path.backup: \_no\_skin

Before starting the application, do some changes in the services.dart file in Code\devote\_app\lib. The change needs to be made in the node URL through which the application will interact with the API. Put the URL accordingly (e.g. localhost:4000 in this case)



**Application UI:**

