**Frontrunning test explained**  
  
Frontrunning is the act of placing a transaction in a queue with the knowledge of a future transaction. On the Ethereum blockchain, front running can occur when bots are able to quote a higher gas price than a pending trade, thus, hastening its processing. Other parties capable of front running are full node operators, which are parties tasked with keeping an eye on network activities, hence, have knowledge of unconfirmed transactions. Front-running is a pervasive issue on public blockchains such as Ethereum. The best remediation is to remove the benefit of front-running in your application, mainly by removing the importance of transaction ordering or time.  
  
In our dao smart contract we avoid the risk of fruntrunning by defining incremental counter inside of smart contract so each proposal would has unique id that is produced by smart contract so each proposal has it’s own ID and if two proposals are similar this ID make them different. So even if the front runner try to create a proposal with the same data, it would has different proposal Id for that and frontrunner would not take any benefit from this frontrunning action.

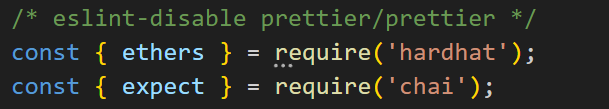
**Tools**

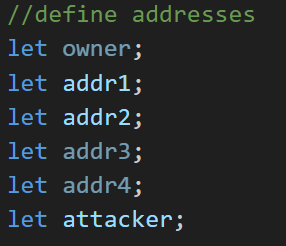
**Hardhat:** Hardhat is an Ethereum development environment for professionals by Nomic Foundation. It is a flexible, extensible, and fast Ethereum development environment that helps developers manage and automate the recurring tasks inherent to developing smart contracts and dApps. Hardhat provides tools such as Hardhat Runner, Hardhat Network, and Hardhat VSCode to help developers easily deploy their contracts, run tests, and debug Solidity code without dealing with live environments. Hardhat is designed to be extended and has all the utilities you need to address your project-specific needs. It also has a composable ecosystem of plugins that add functionality and integrate your existing tools into a smooth workflow. (<https://hardhat.org/>)  
We use hardhat to test frontrunning process to see the result that even frontrunner take the user transaction and put his transaction sooner he won’t achieve any benefits from this trick.  
  
**Javascript:** JavaScript is a programming language that is commonly used to add interactivity and other dynamic features to a software. It is the world’s most popular programming language and is the programming language of the Web. JavaScript is easy to learn and can be used for a wide range of tasks, from simple tasks like changing the content of a web page to more complex tasks like creating interactive games and animations. We use this language to write functions of frontrunning attack.

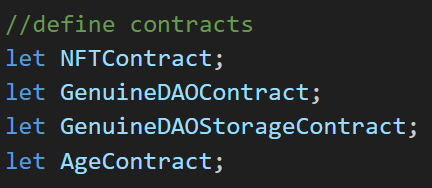
**Nodejs**:Node.js is an open-source, cross-platform JavaScript runtime environment1. It is built on Chrome’s V8 JavaScript engine and is used to execute JavaScript code outside of a web browser. Node.js allows developers to use JavaScript to write command line tools and server-side scripts, making it possible to develop full-stack applications using only JavaScript. You can download Node.js from their official website (<https://nodejs.org/en>)  
  
**Chai:** Chai is a BDD (Behavior Driven Development) and TDD (Test Driven Development) assertion library for Node.js and blockchain that can be paired with any JavaScript testing framework like hardhat. It provides a chain-capable BDD style that provides an expressive language and readable style, while the TDD assert style provides a more classical feel. Chai has several interfaces that allow the developer to choose the most comfortable one. It also has a composable ecosystem of plugins that add functionality and integrate existing tools into a smooth workflow. Chai is available on npm and can be installed by node package manager.( [https://www.chaijs.com/](https://www.chaijs.com/)) )  
  
**Ethers**: The ethers library is a complete and compact library for interacting with the Ethereum Blockchain and its ecosystem. It was originally designed for use with ethers.io and has since expanded into a more general-purpose library. The library provides features such as keeping your private keys in your client, importing and exporting JSON wallets, importing and exporting BIP 39 mnemonic phrases, and connecting to Ethereum nodes over JSON-RPC, INFURA, Etherscan, Alchemy, Cloudflare or MetaMask. ENS names are first-class citizens in the ethers library; they can be used anywhere an Ethereum address can be used. The ethers library is tiny, has complete functionality for all your Ethereum needs, has extensive documentation, a large collection of test cases which are maintained and added to, is fully TypeScript ready, and is MIT licensed (including ALL dependencies). (<https://docs.ethers.org/v5/>)

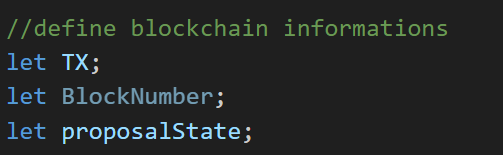
**Code explain**

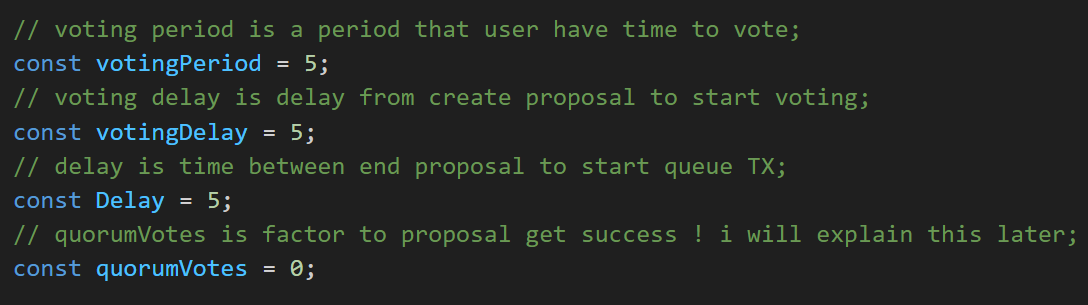
In the path (/test/frontRunning.js) we have our frontrunning test file.  
  
In first lines we import libraries that we need (hardhat and chai). These two libraries help us to test frontrunning. Also hardhat itself contains ethers that we use to work with contracts.



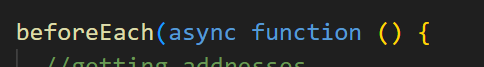
Then we create chai describe with contains our test codes: (<https://hardhat.org/tutorial/testing-contracts>)  
  
  
Then we define javascript variables to store users and contracts information:  


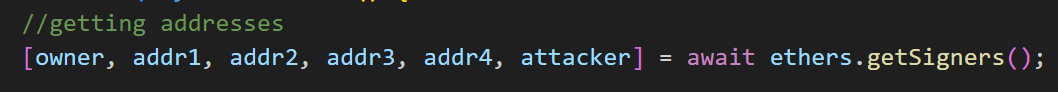


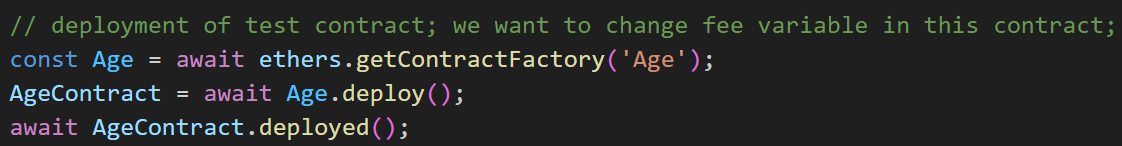




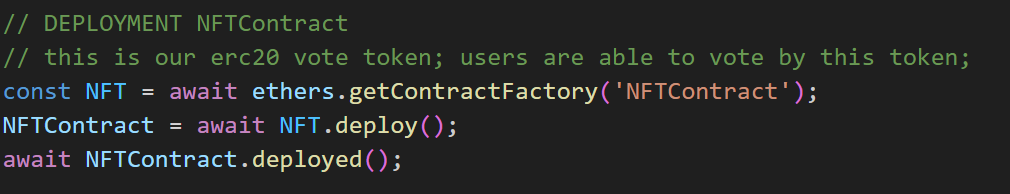
In beforeach function we deploy main contracts and initialize users (<https://hardhat.org/tutorial/testing-contracts>)

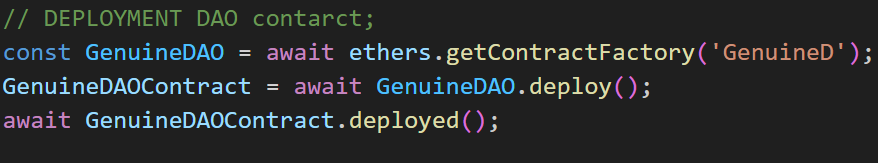


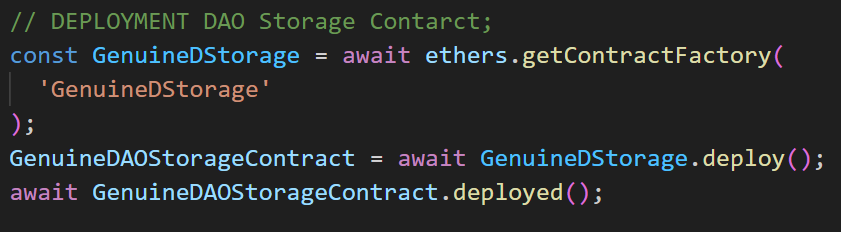
Here we get addresses from ethers  


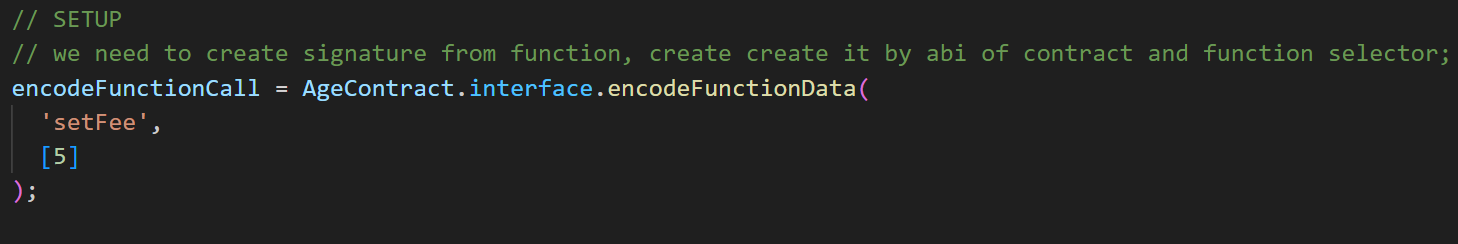
Here we deploy age contract that contains fee percent that we want to change with dao:  


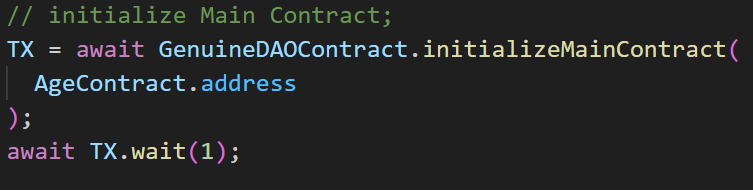
Deploying NFT contract that is voting token that users can vote to proposals using this token:

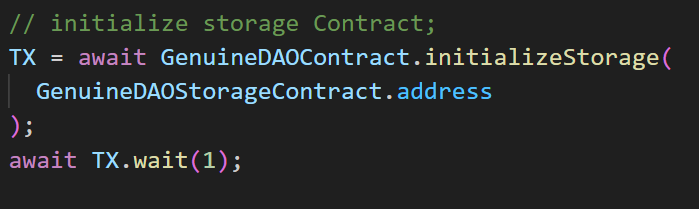


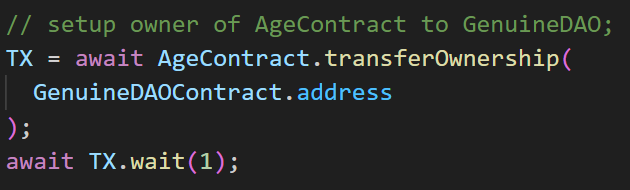
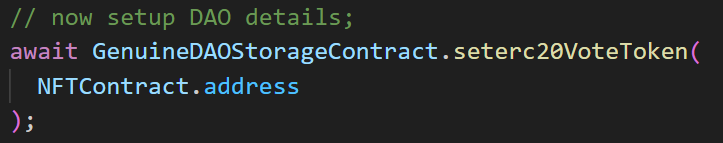
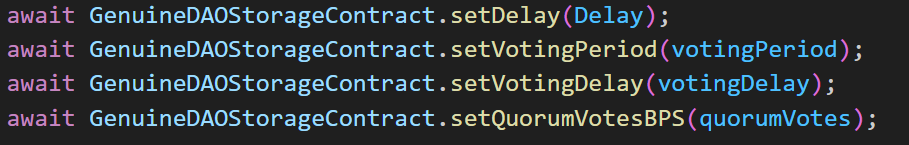
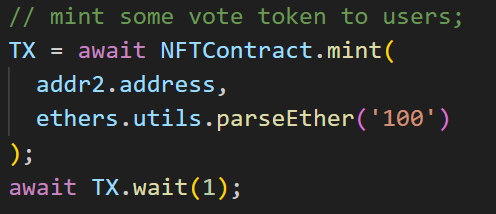
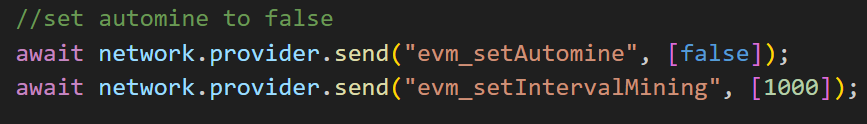
Deploying Dao contract that is has functions for creating proposals and voting:  


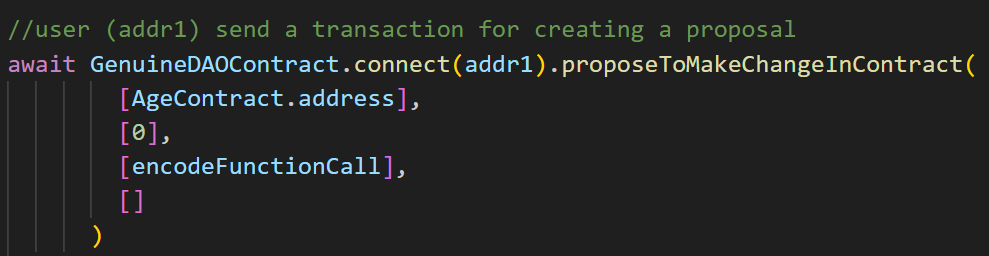
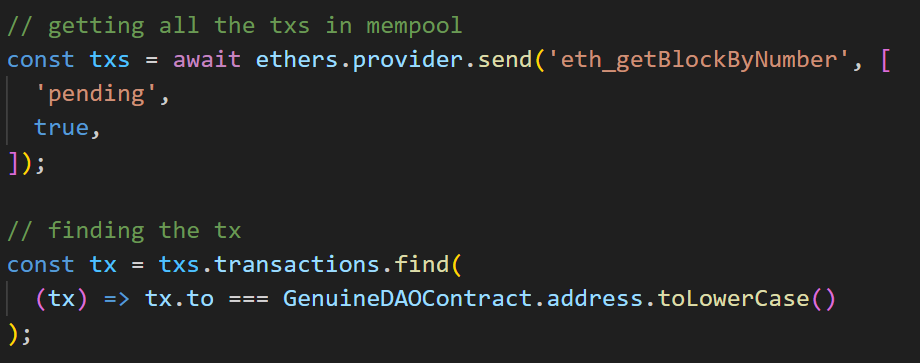
Deploying Dao Storage contract that plays the role of data storage for our main dao contract:  


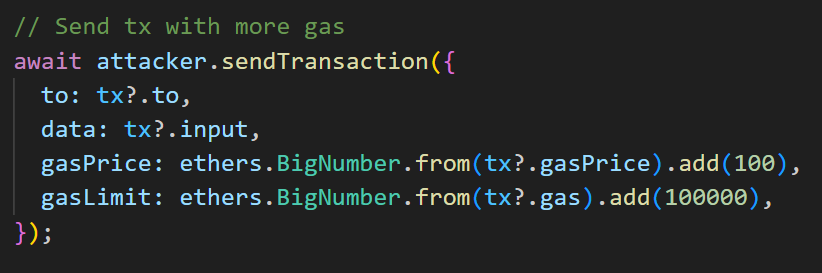
Here we create an a signature that is necessary for creating proposals later:  


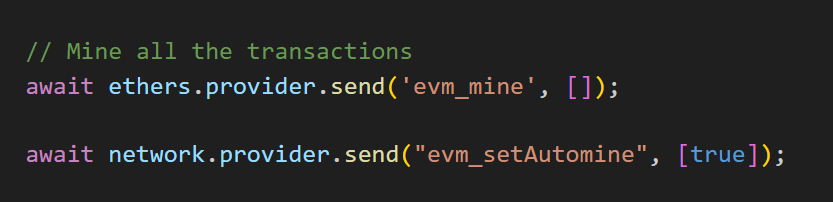
After deploying contracts we need to link them together when we are initializing them, here we set age contract as main contract for our Dao so dao can know which contract is the main contract to change functions of that:  


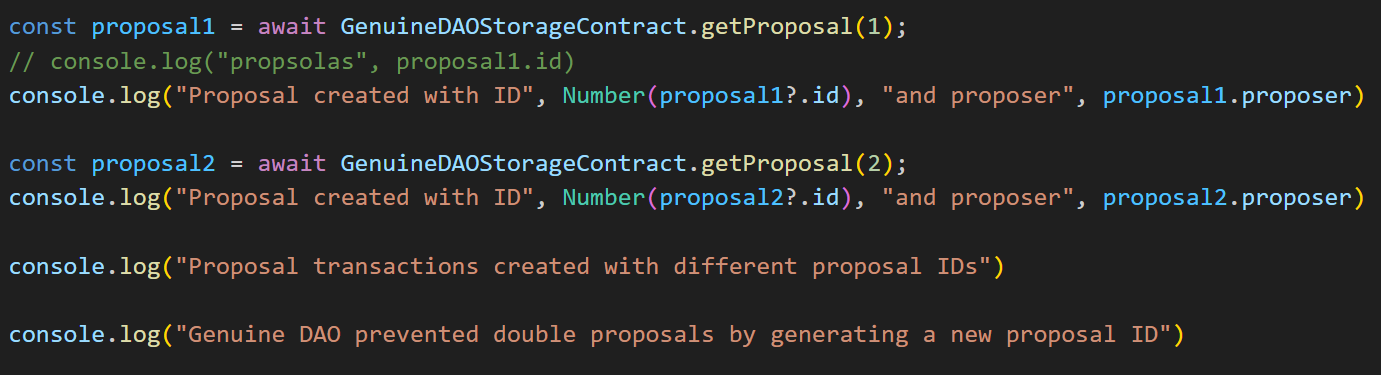
Also we need to link our storage contract to our Dao contract:  


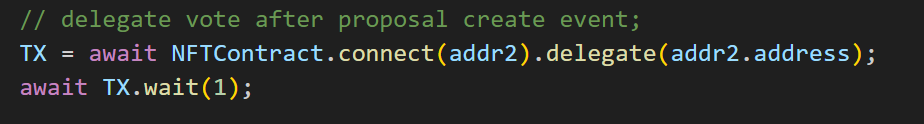
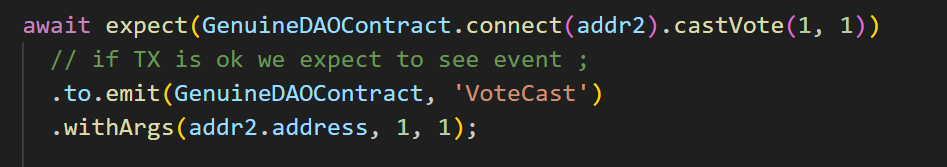
The important action that we need to do is changing owner of Age contract to Dao contract so the previous owner won’t be able to change fee because now Dao contract is the owner and The fee can only be changed by proposals.  
  
  
Then we need to link our deployed NFT contract as voting token to Dao contract:  
  
  
At the end of initializations we need to set voting period and delay for Dao storage contract:  
  
  
Here we mint 100 NFT token (as vote token) for address2 so this address can vote to proposals later:  
  
  
  
Here we implement blockchain mining time:  


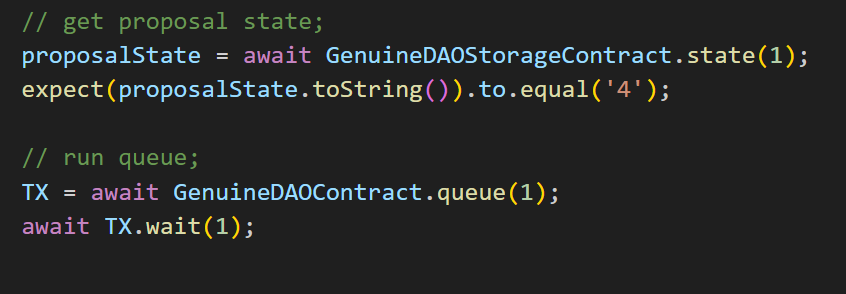
Here user with addr1 create a transaction fro making a proposal with signature:  
  
  
  
addr1 transaction goes in the mempool and waits for miners to mine it. Here attacker looks into mempool and find addr1 pending proposal transaction and pick it:  


Then attacker sends a transaction with the same data but by more gas price so miner, mine attacker transaction first because it offers more gas price:  




After both transactions been mined (attacker proposal mined first because of higher gas price offered). But we see that we don’t have doubled proposal actually we have two proposals with different proposal ID:  


However attacker made proposal first but there is no benefit for him because he has a different proposal ID.  
  
Then in voting time voter can delegate and vote to the proposals:  
  


After voting ended we can see proposal state and queue transactions:  


At the end we can execute the proposal and see that fee is changed by voted proposal:  
