**PRACTICAL: 3**

**AIM:** Code Your Own Cryptocurrency on Ethereum (ERC-20 Token).

**Course:**

[**https://www.udemy.com/course/code-your-own-cryptocurrency/?srsltid=AfmBOopjvZsDJ6uJbYHew5nbOgyWiNRw0w2-x5fe6z\_D53kyYdmjOQA3**](https://www.udemy.com/course/code-your-own-cryptocurrency/?srsltid=AfmBOopjvZsDJ6uJbYHew5nbOgyWiNRw0w2-x5fe6z_D53kyYdmjOQA3)

**THEORY:**

ERC-20 is the technical standard for fungible tokens created using the Ethereum blockchain. A fungible token is one that is exchangeable with another token, whereas the well-known ERC-721 non-fungible tokens (NFTs) are not.

ERC-20 allows developers to create smart-contract-enabled tokens that can be used with other products and services. These tokens are a representation of an asset, right, ownership, access, cryptocurrency, or anything else that is not unique in and of itself but can be transferred.

Smart contracts were becoming more popular in 2015, but several issues needed to be addressed. One of the most pressing was that anyone could make a token, but they were not always interoperable with other tokens. Without a standardized token methodology, there wasn't a way to ensure that all the different tokens could be created, used, or exchanged by everyone using the blockchain.

ERC-20 was proposed by developer Fabian Vogelsteller in 2015 to address the need for a standard within smart contracts on the Ethereum blockchain. Vogelsteller submitted the proposal via the project's Github page as an Ethereum Request for Comment (ERC). As it was the twentieth comment, it was assigned the designation ERC-20.

Following the procedure used by the Ethereum developer community at the time, the proposal was approved and implemented in 2017 as Ethereum Improvement Proposal 20 (EIP-20). However, it is still called ERC-20 because that's how it was known until it was approved.

Because the request was approved and implemented, smart contract tokens implemented on the Ethereum blockchain must conform to this standard if the developers want them to be interchangeable and advertise that their token is ERC-20 compliant.

ERC-20 is a list of functions and events that must be implemented into a token to be considered ERC-20 compliant. These functions (called methods in the ERC) describe what must be included in the smart-contract-enabled token, while events describe an action. The functions a token must have been:

* **TotalSupply**: The total number of tokens that will ever be issued
* **BalanceOf**: The account balance of a token owner's account
* **Transfer**: Automatically executes transfers of a specified number of tokens to a specified address for transactions using the token
* **TransferFrom**: Automatically executes transfers of a specified number of tokens from a specified address using the token
* **Approve**: Allows a spender to withdraw a set number of tokens from a specified account, up to a specific amount
* **Allowance**: Returns a set number of tokens from a spender to the owner

The events that must be included in the token are:

* **Transfer**: An event triggered when a transfer is successful
* **Approval**: A log of an approved event (an event)

The following functions are optional and are not required, but they enhance the token's usability:

* Token's name (optional)
* Its symbol (optional)
* Decimal points to use (optional)

These functions provide a common structure for tokens so that they can be easily accessed, recognized, reviewed, and used. This reduces the confusion users and application developers would have if every smart contract's token had different information contained within it. Additionally, the code functions assist in determining the number of tokens in circulation, storing and returning balances, making transfer and withdrawal requests, granting approval, and agreeing to automated transfers.

**CODE:**

|  |
| --- |
| * npm install -g truffle * npm install -g solc * mkdir token\_sale/ * cd token\_sale * truffle init * echo.> contracts/Trushang.sol * truffle migrate –reset * truffle console * npm install |

**OUTPUT:**

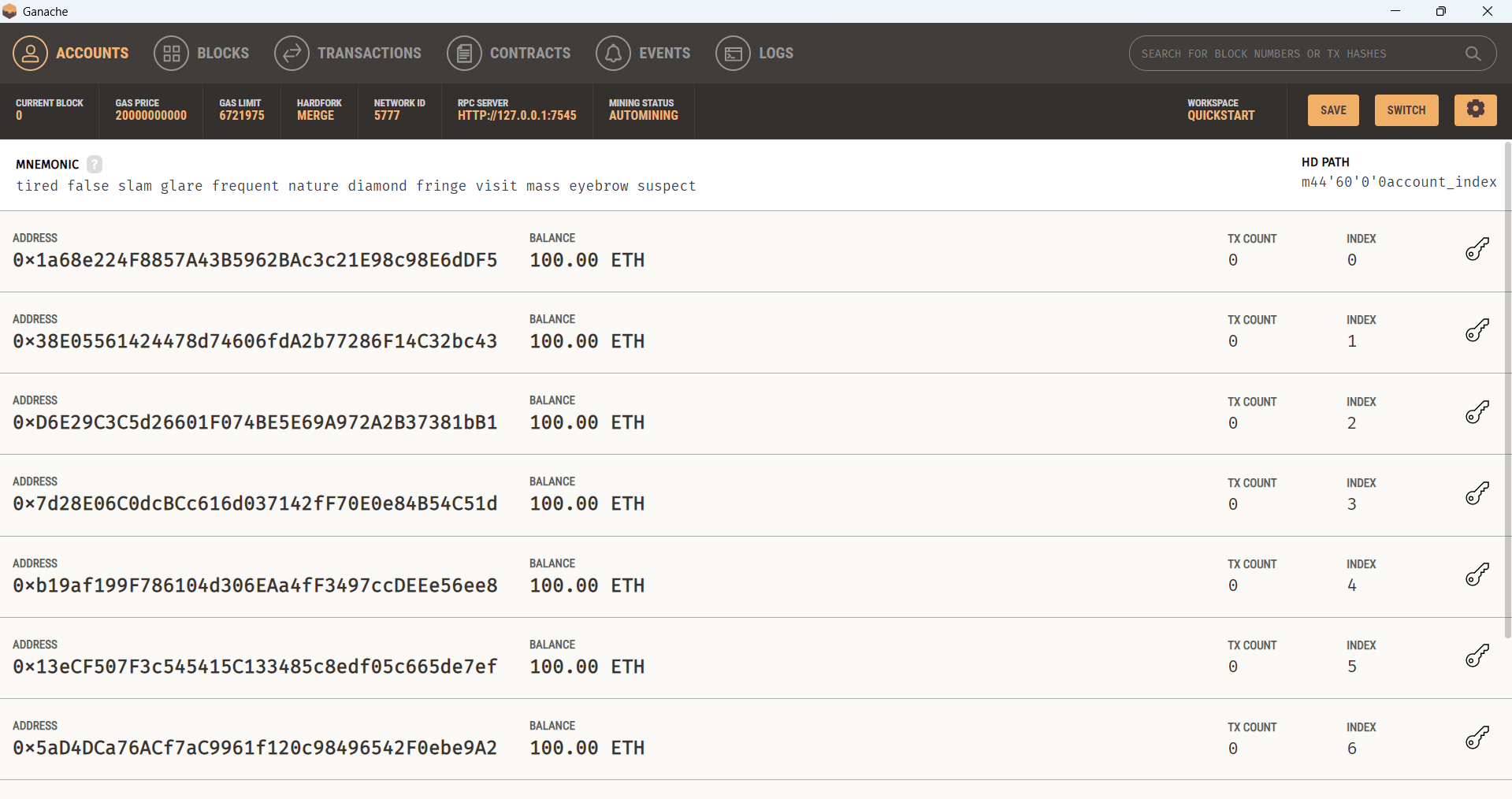


Figure 1:Start Ganache

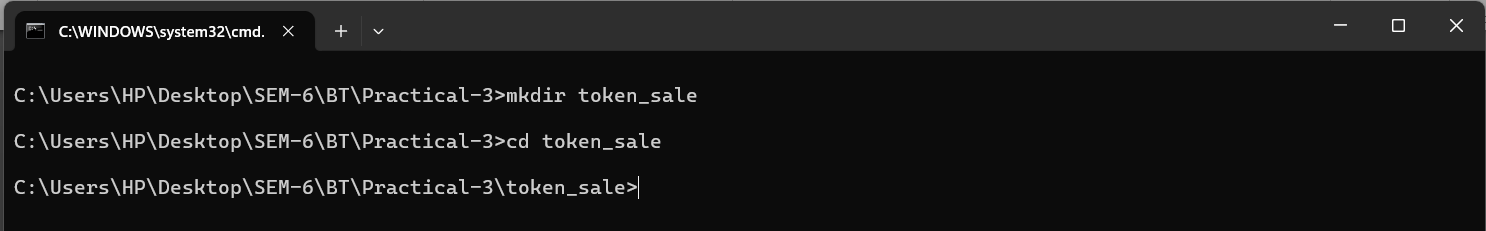


Figure 2:Create project directory

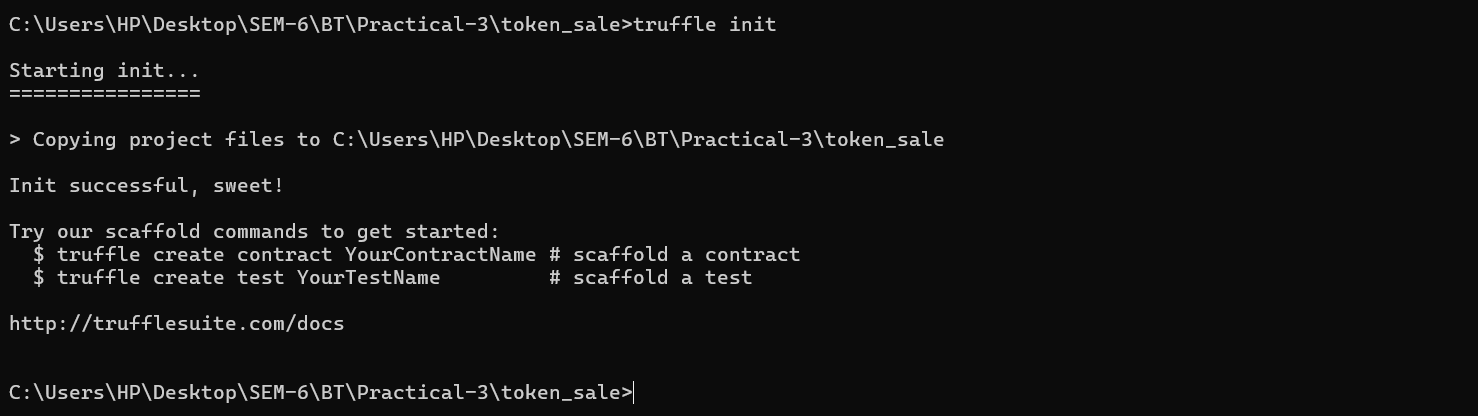


Figure 3:create truffle project in directory

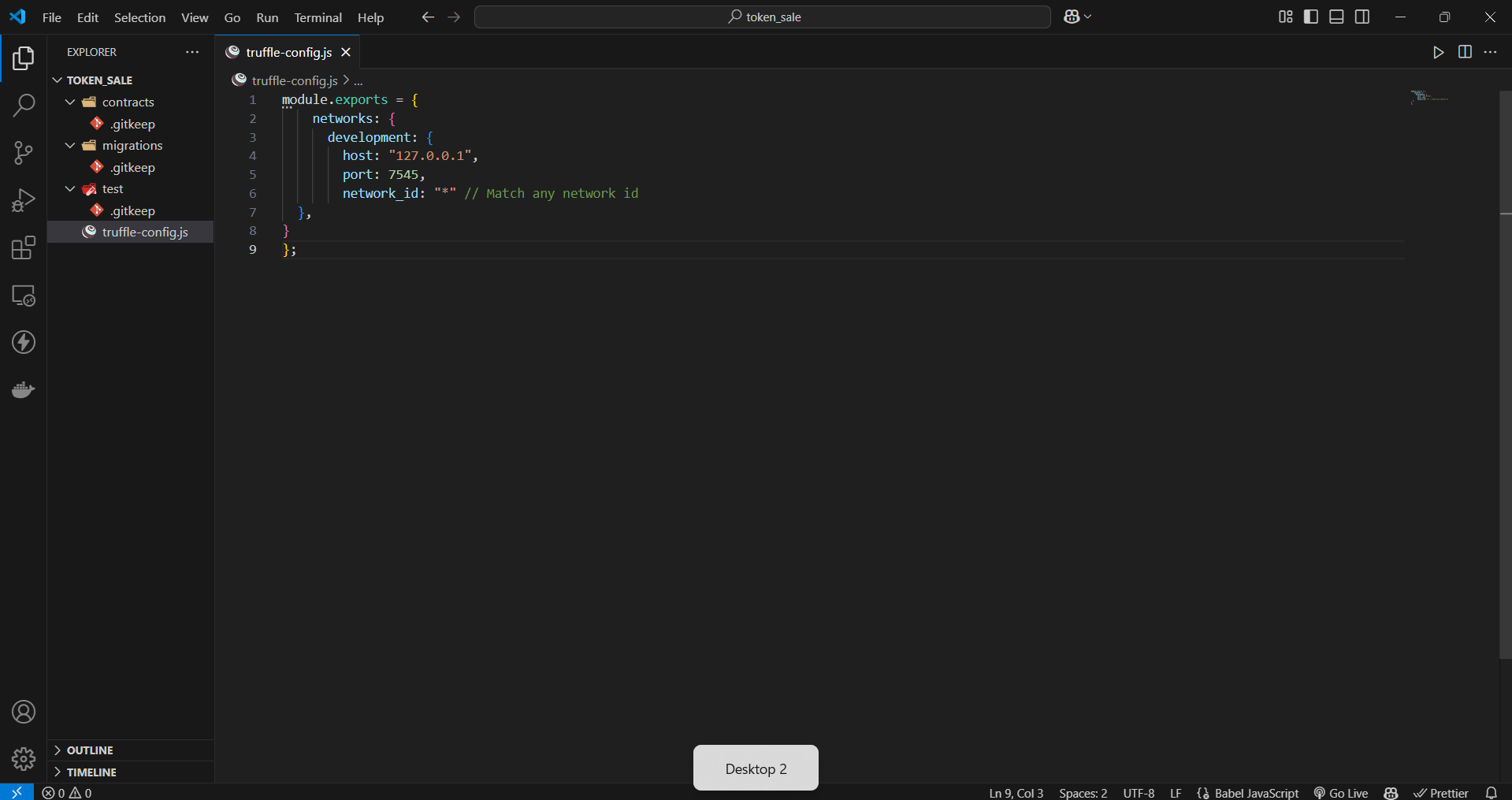


Figure 4:Config your truffle-config.js

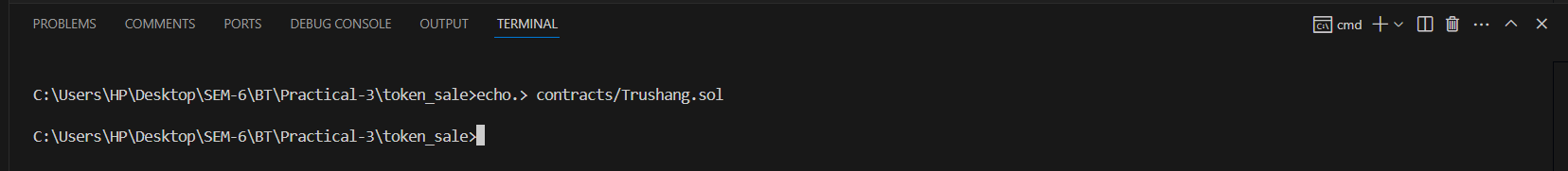


Figure 5:Create a new Contract file

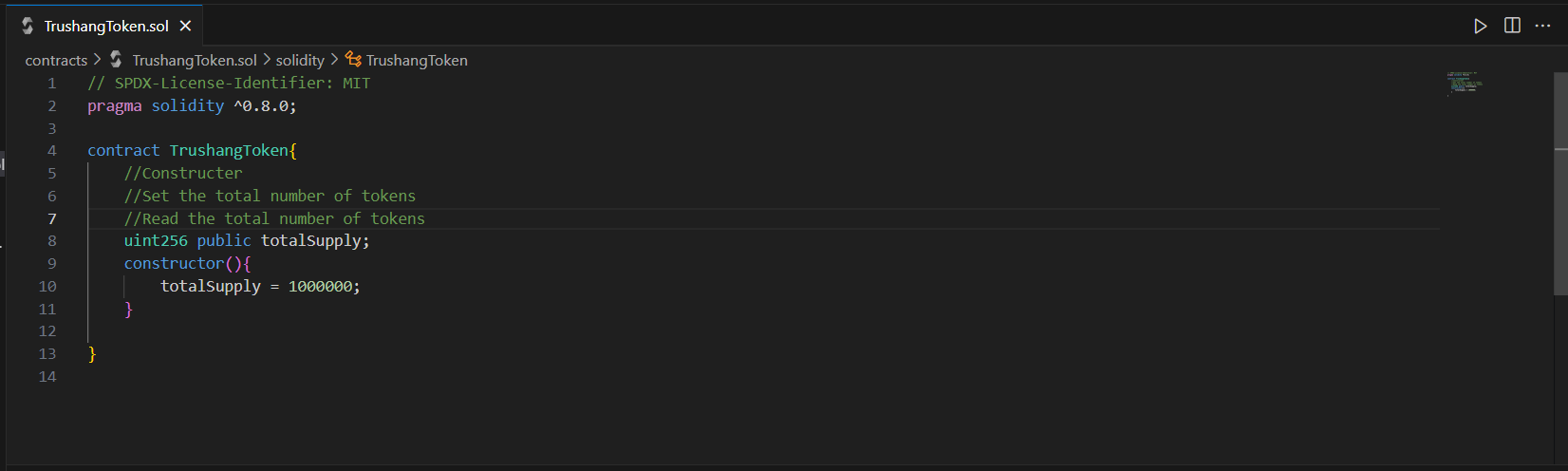
****

Figure 6:Write your Token contract



Figure 7:Create a migration file

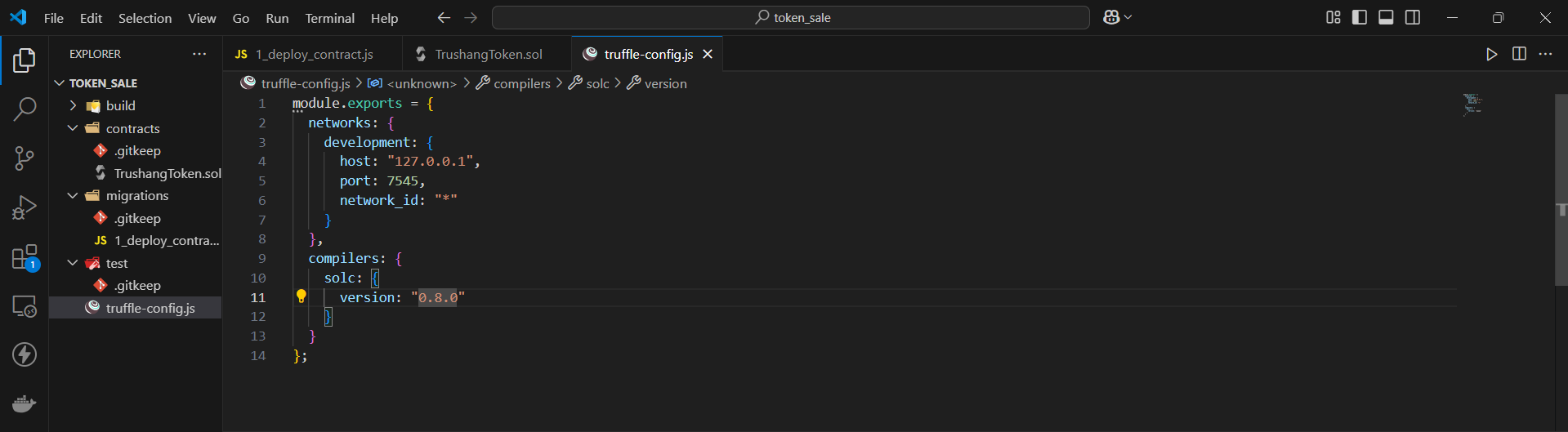


Figure 8:write logic for migration and also update truffle-config.jS

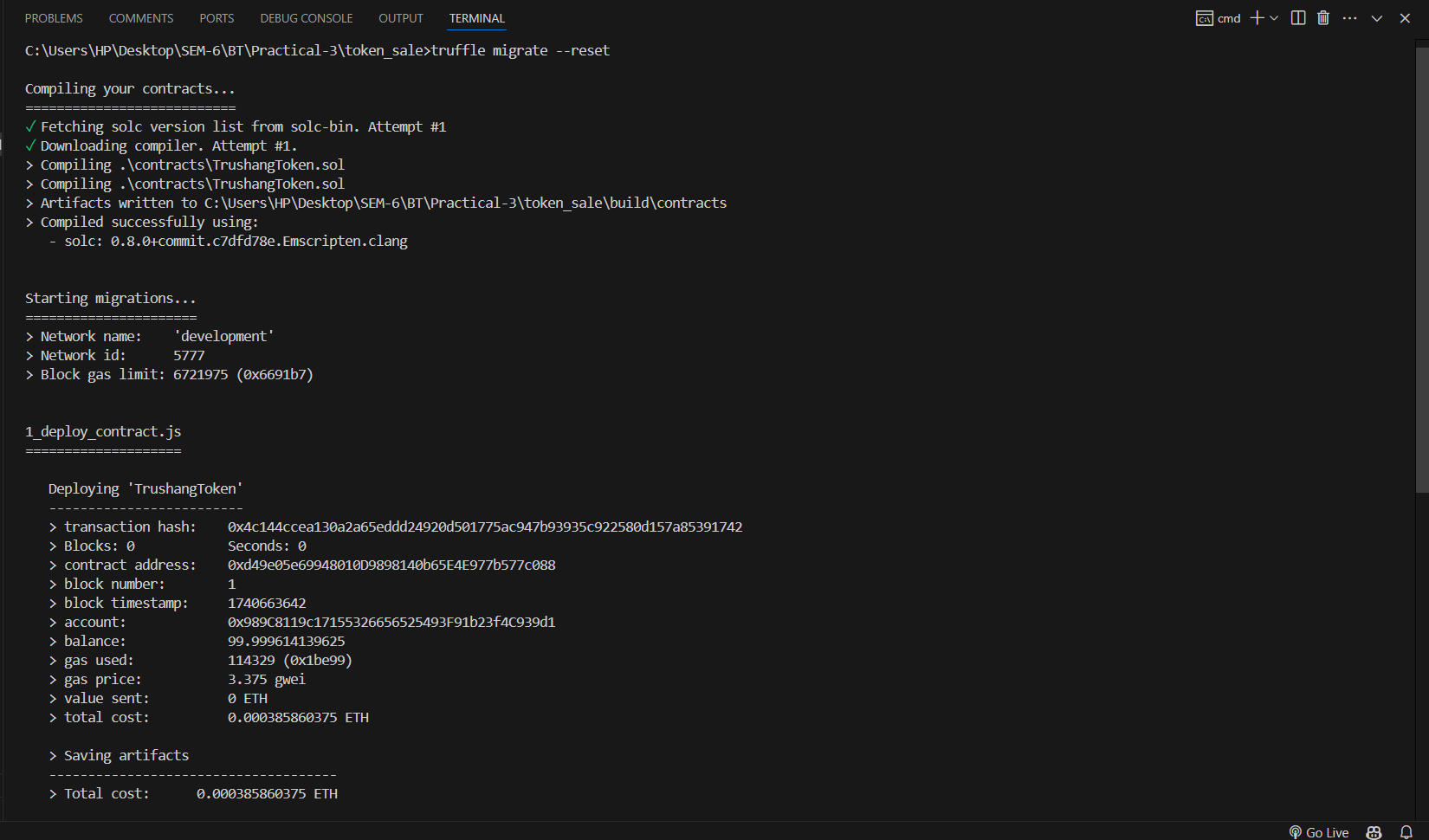


Figure 9:Run migration file

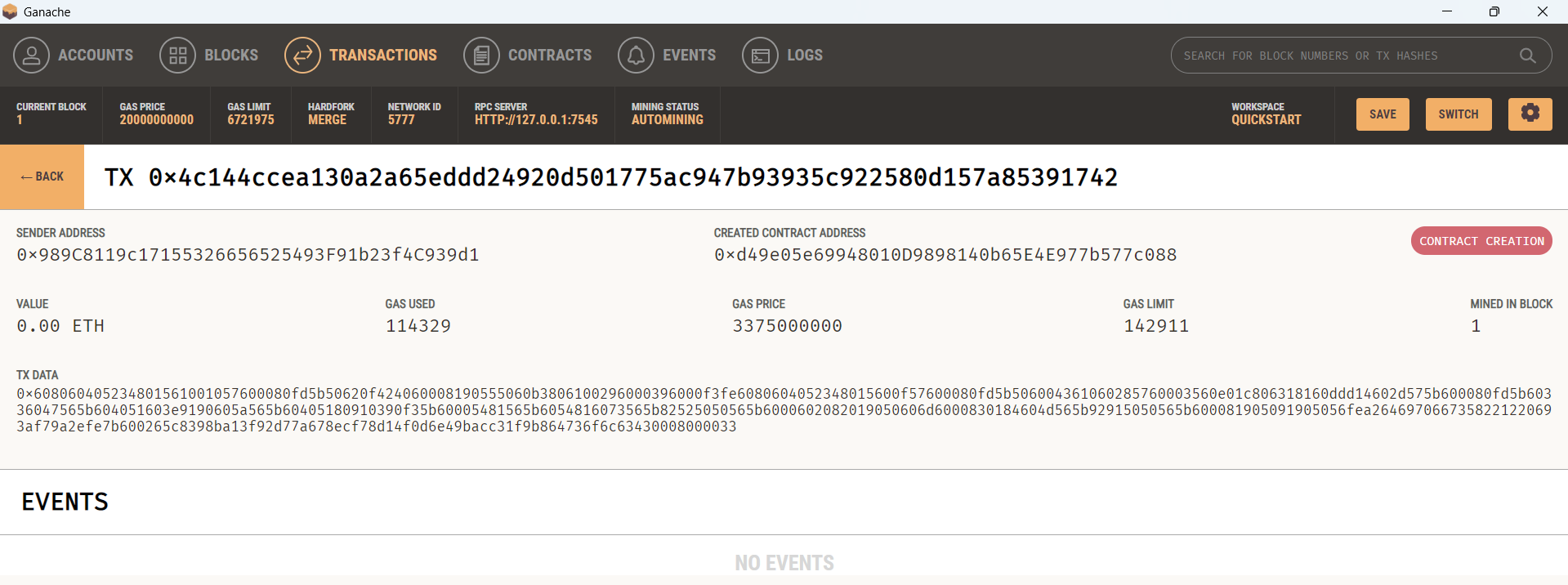


Figure 10:Check Ganache transaction

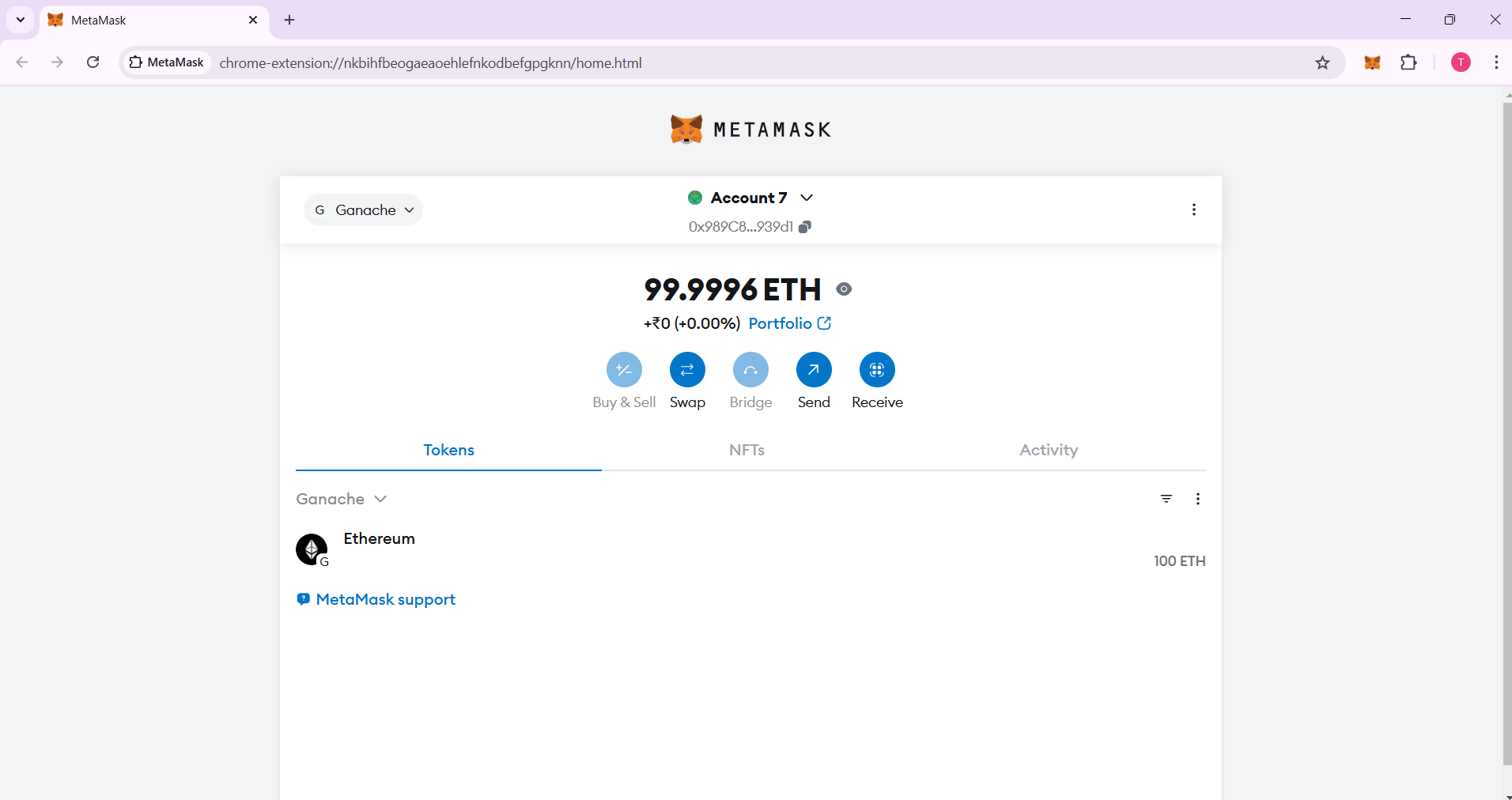


Figure 11:See Account in MetaMask

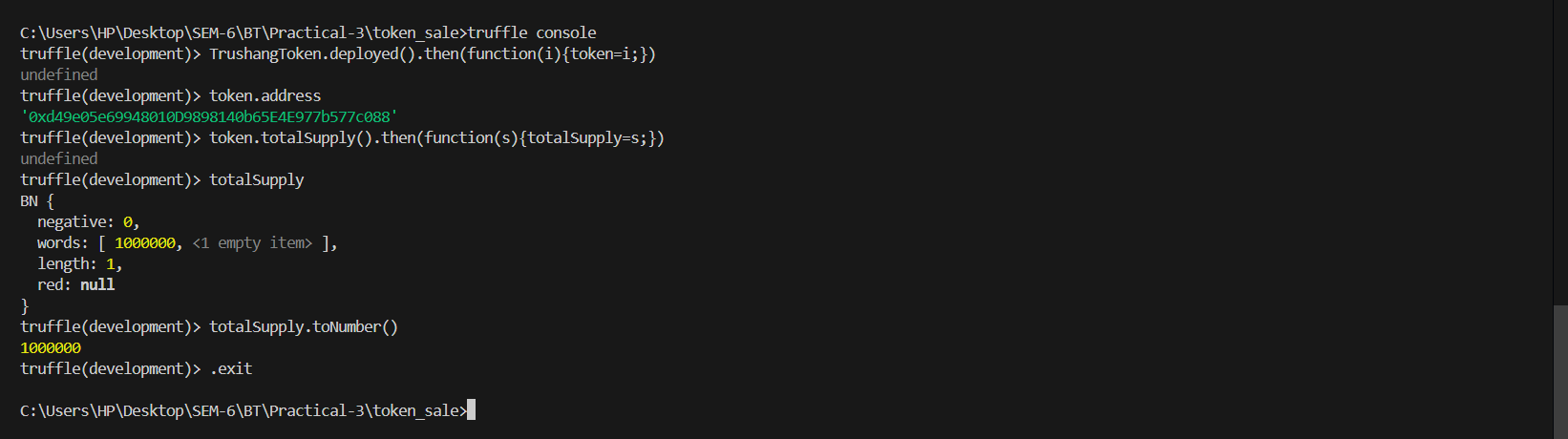


Figure 12:Open truffle console and verifying smart contract are created or not and if created then it’s hash value and ganache contract value

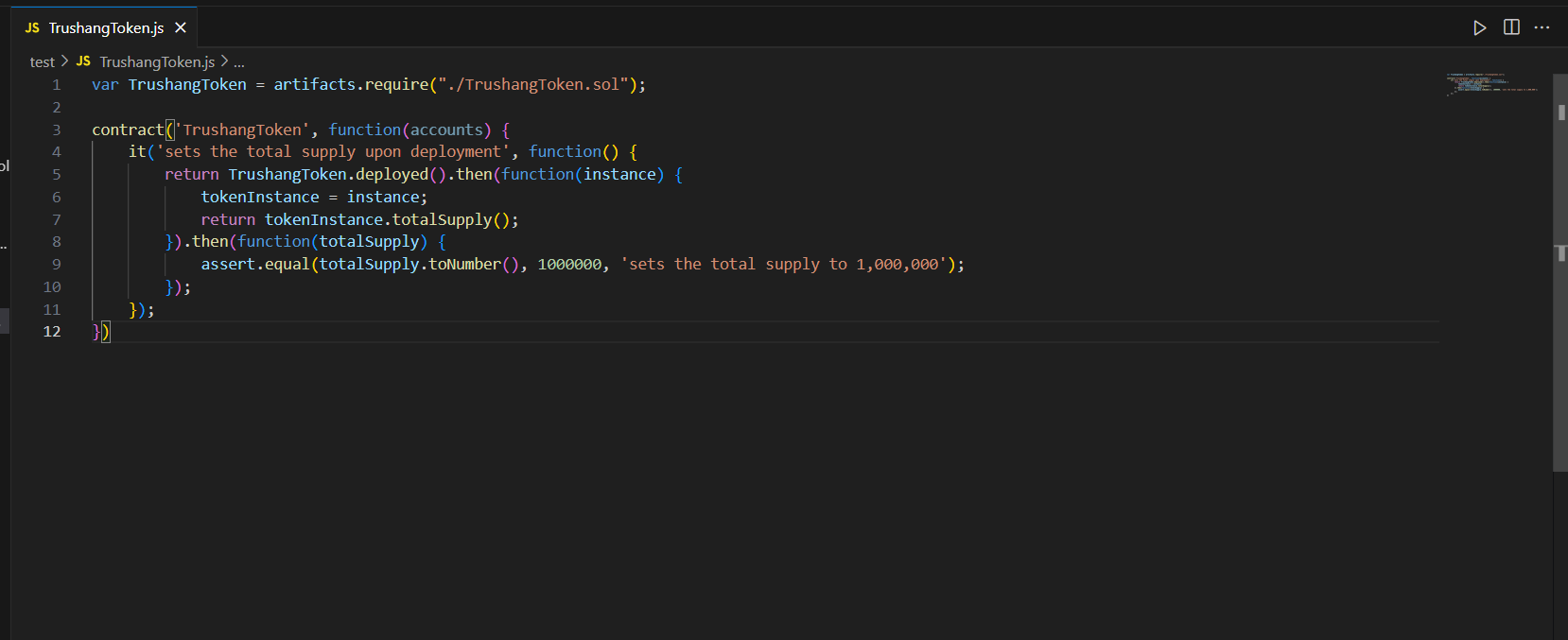


Figure 13: create test/TrushanToken.js file

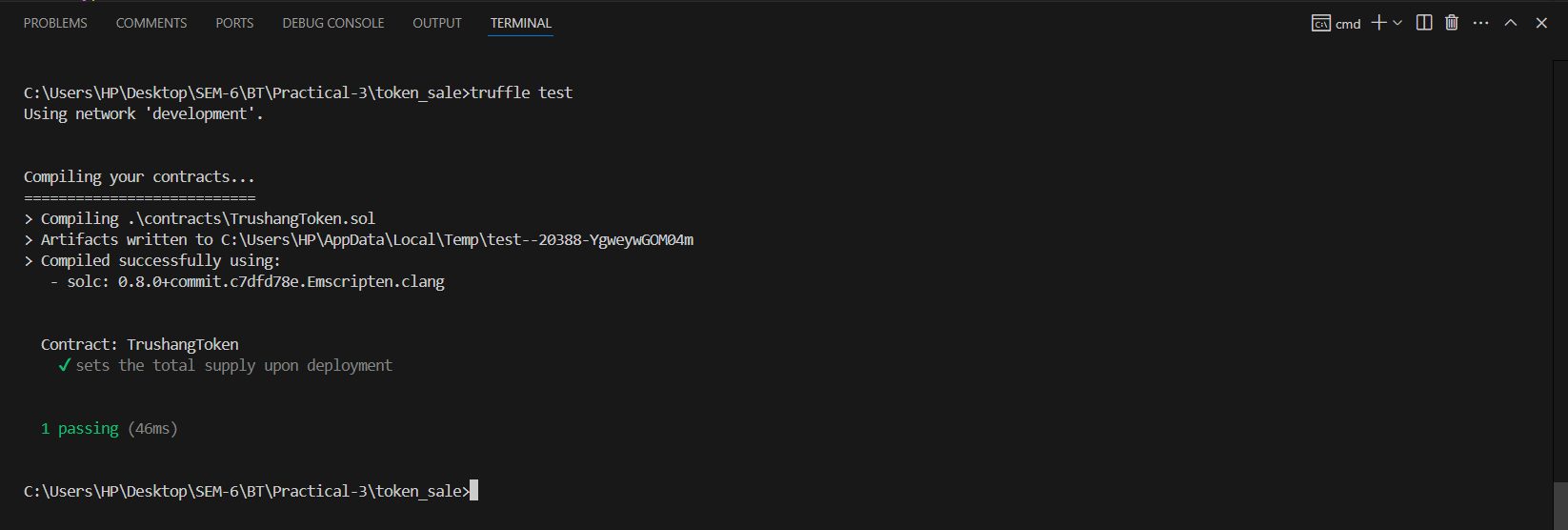


Figure 14:Testing the deployment

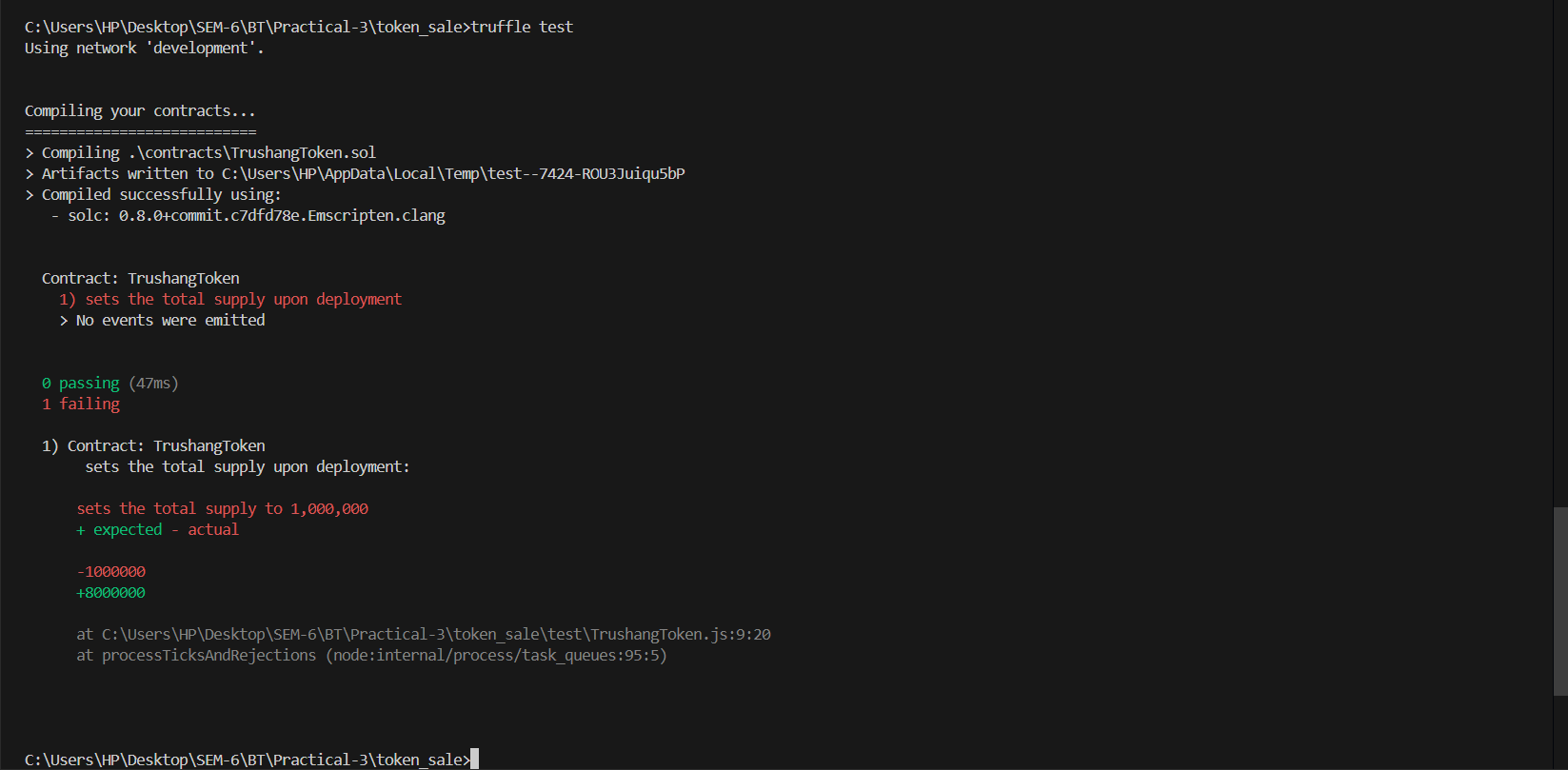


Figure 15:Change the value 1000000 to 800000 and again test it

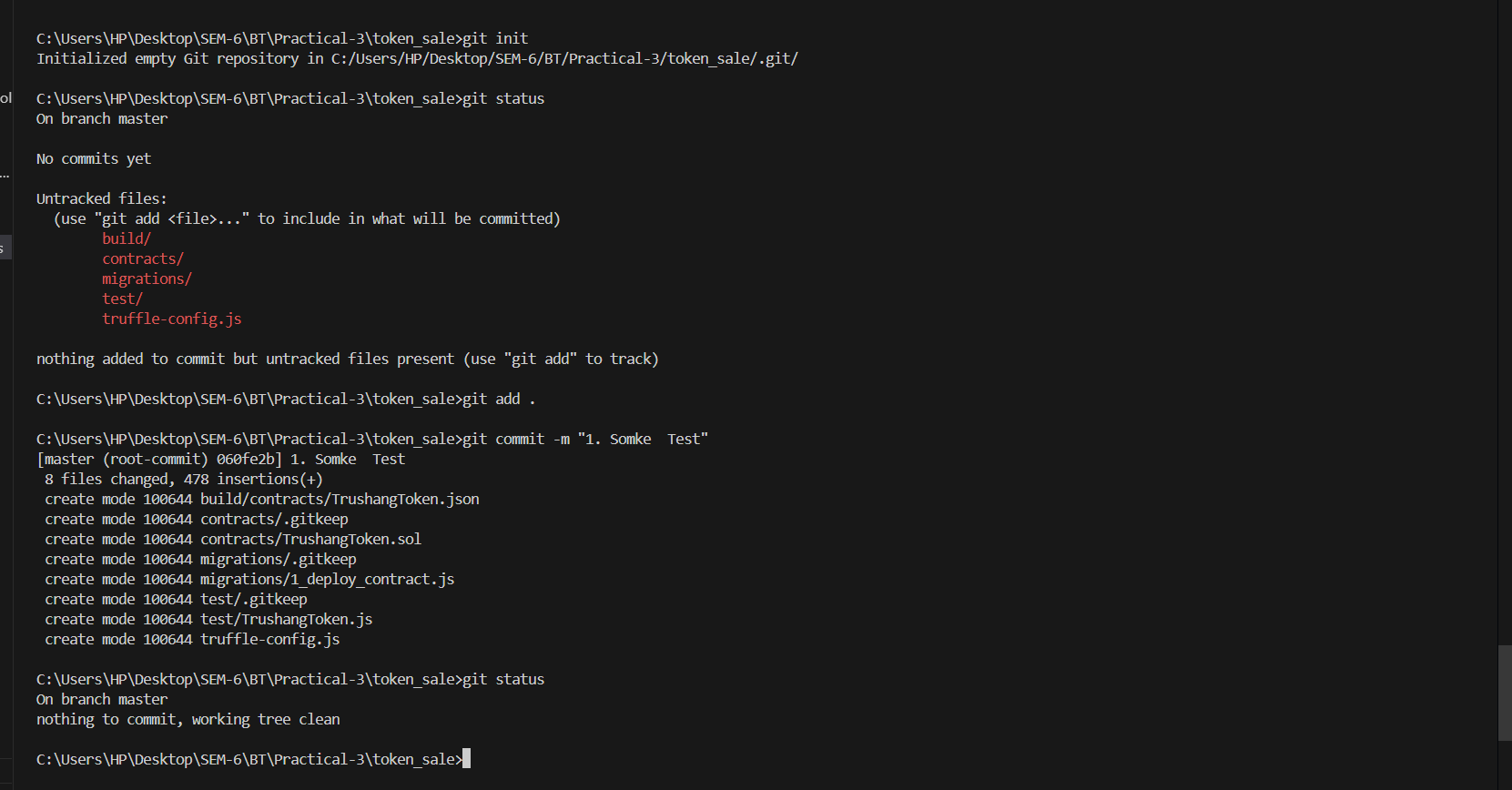


Figure 16:Create git repo and add to the github

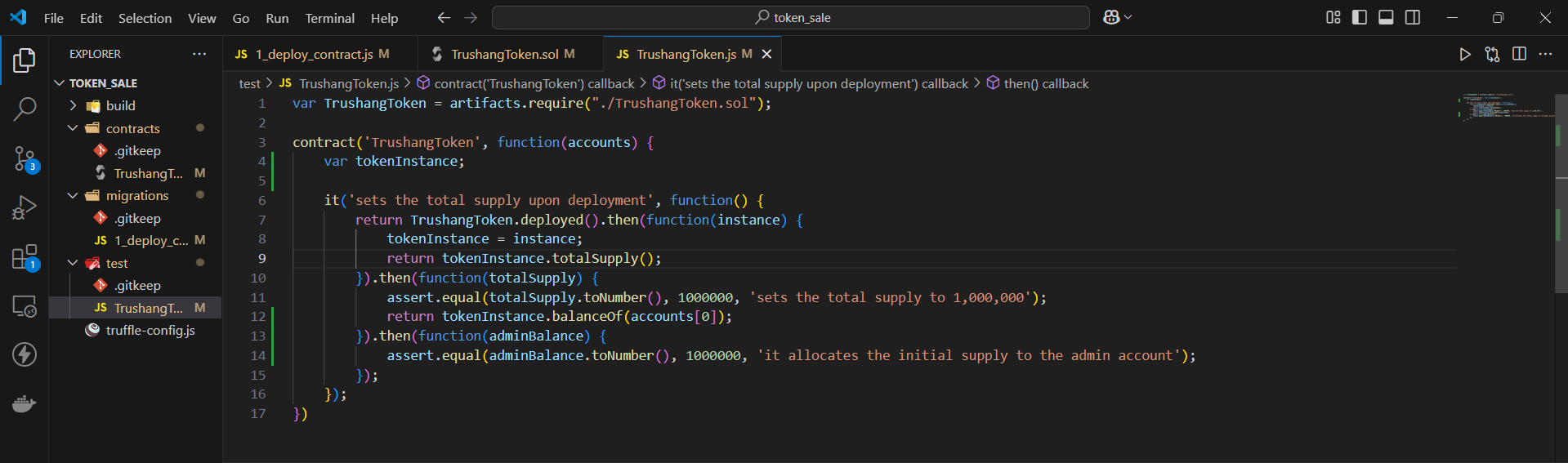


Figure 17: Set initial value 1,000,000

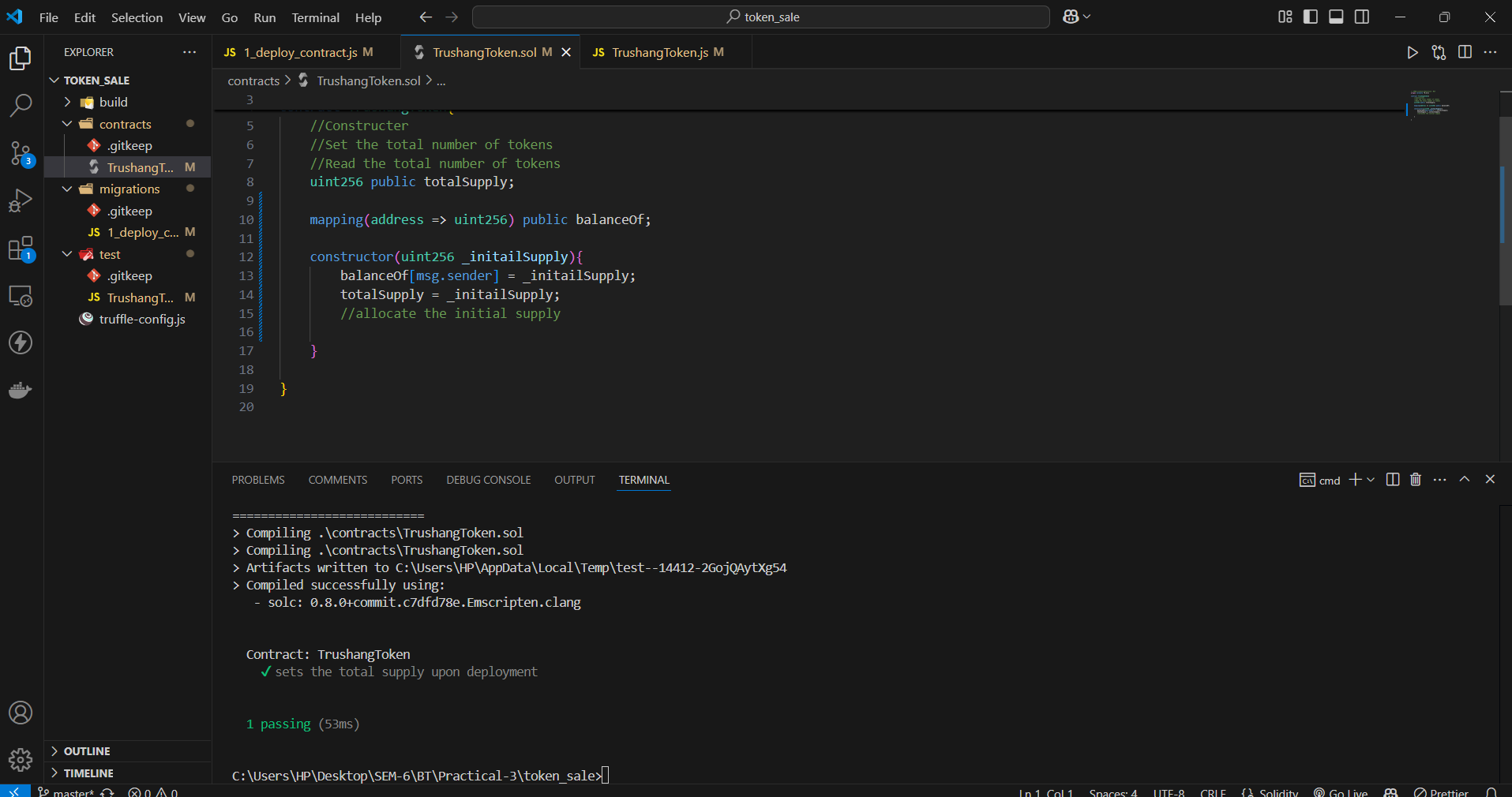


Figure 18:Test the code

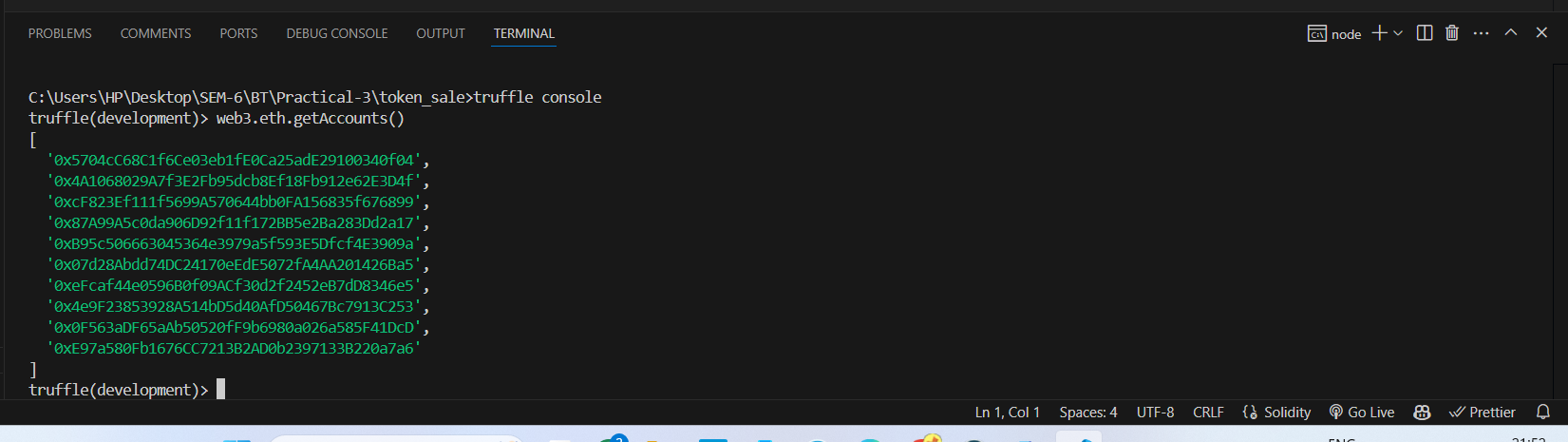


Figure 19:In truffle see how many accounts in our network using web3.eth.getAccounts()

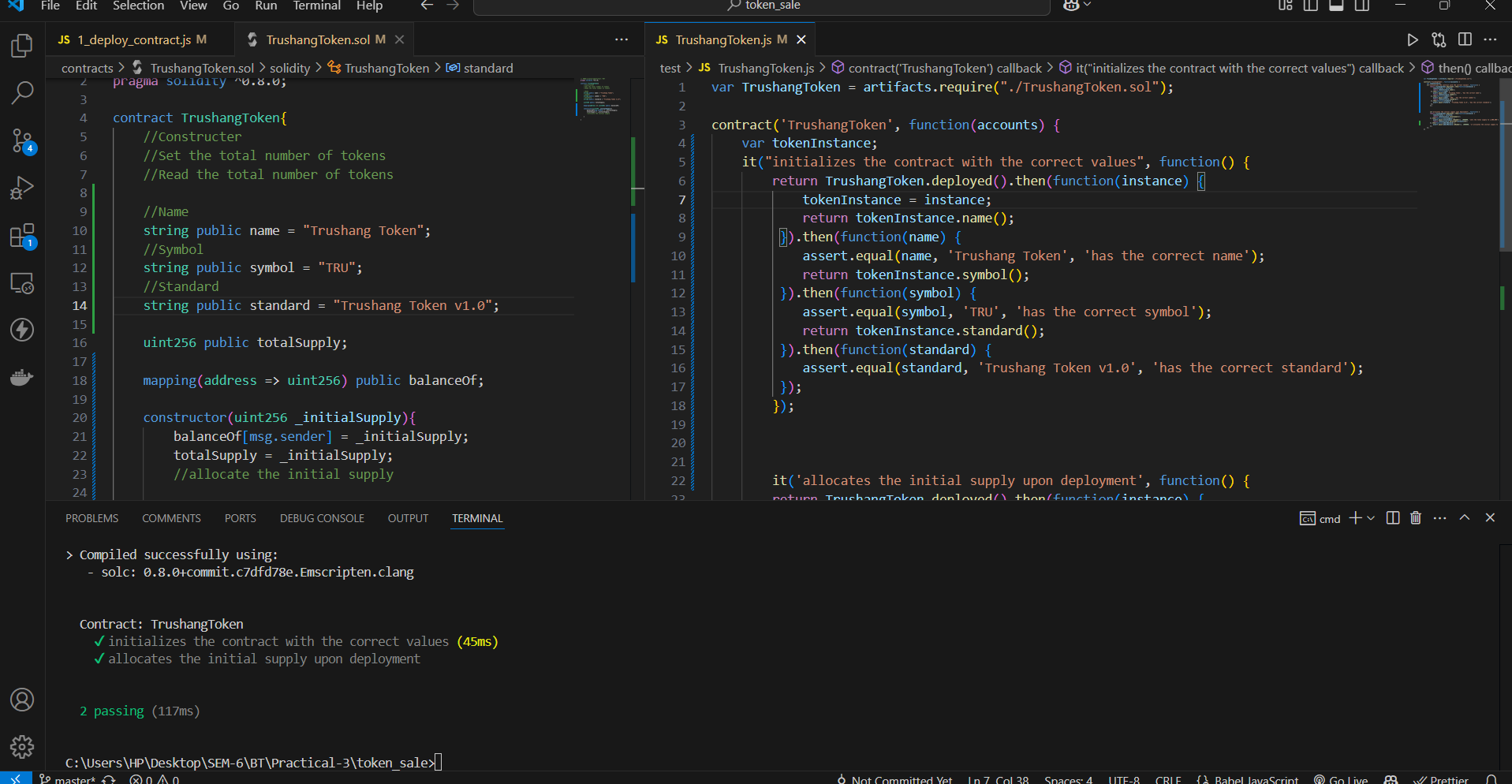


Figure 20:Add token name, symbol, and standard in our code and then test it

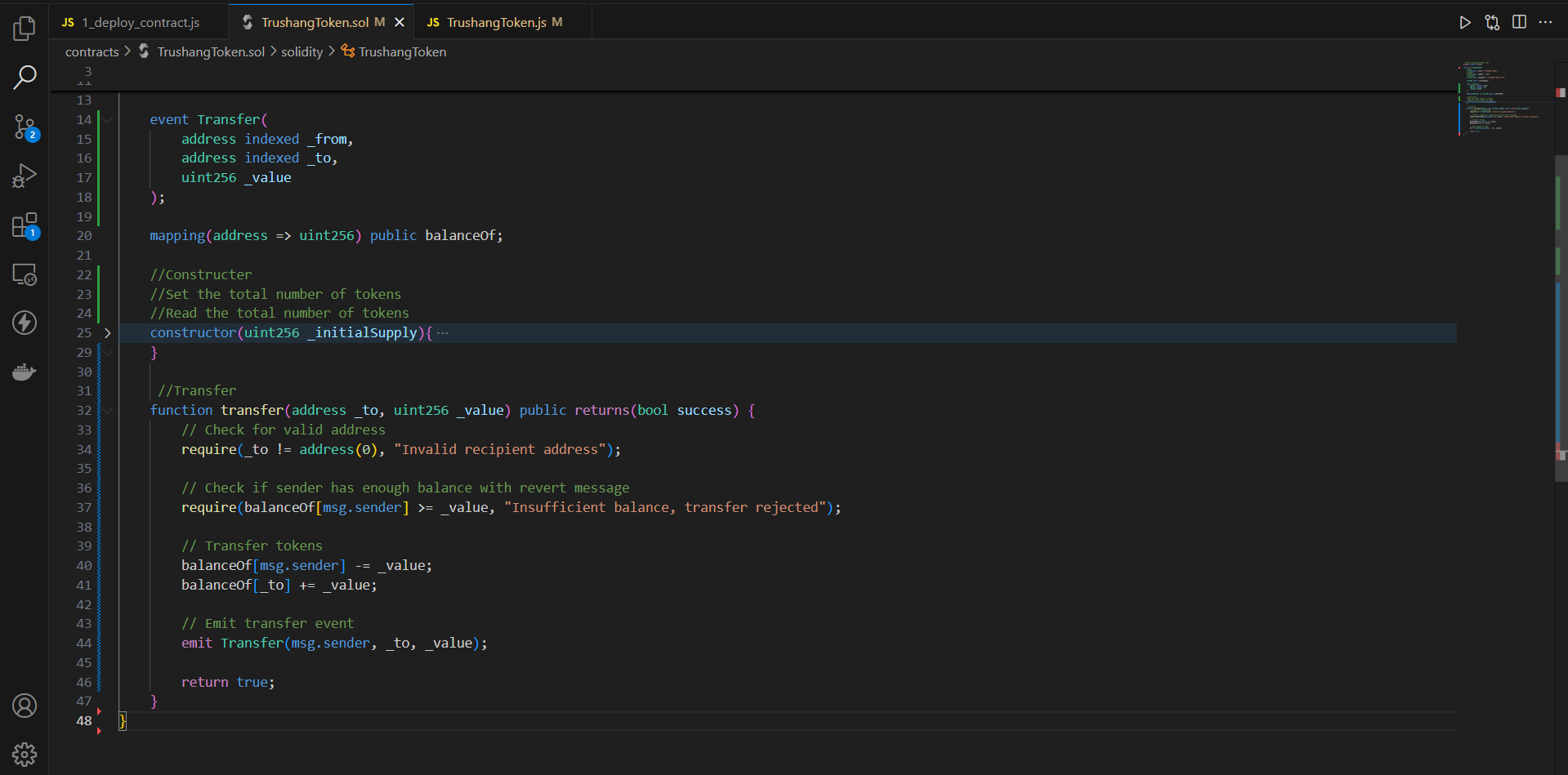


Figure 21:Create Constructure



Figure 22:Create a Token Ownership



Figure 23:Test the contract

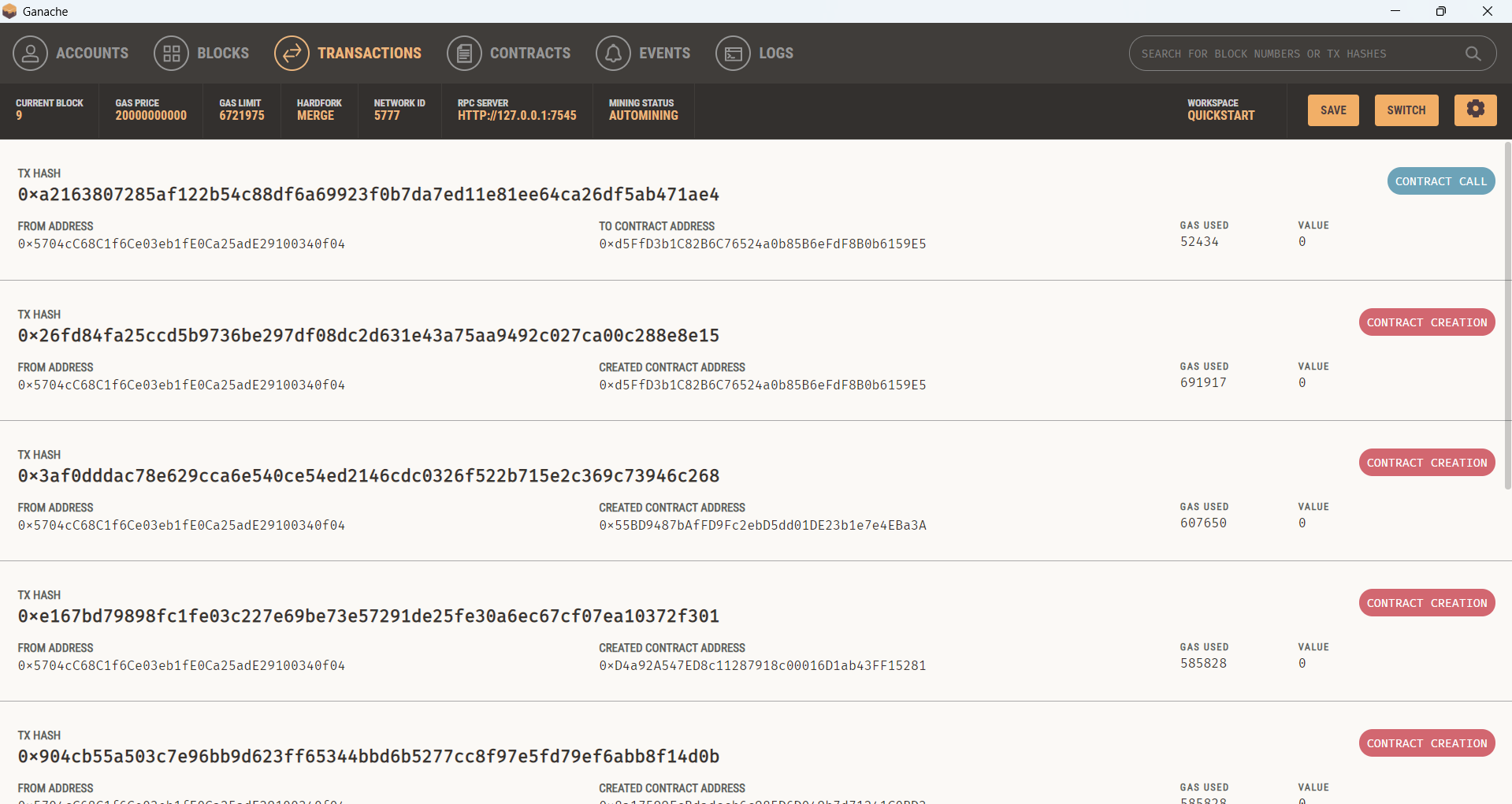


Figure 24:Ganach transaction for ownership transfer

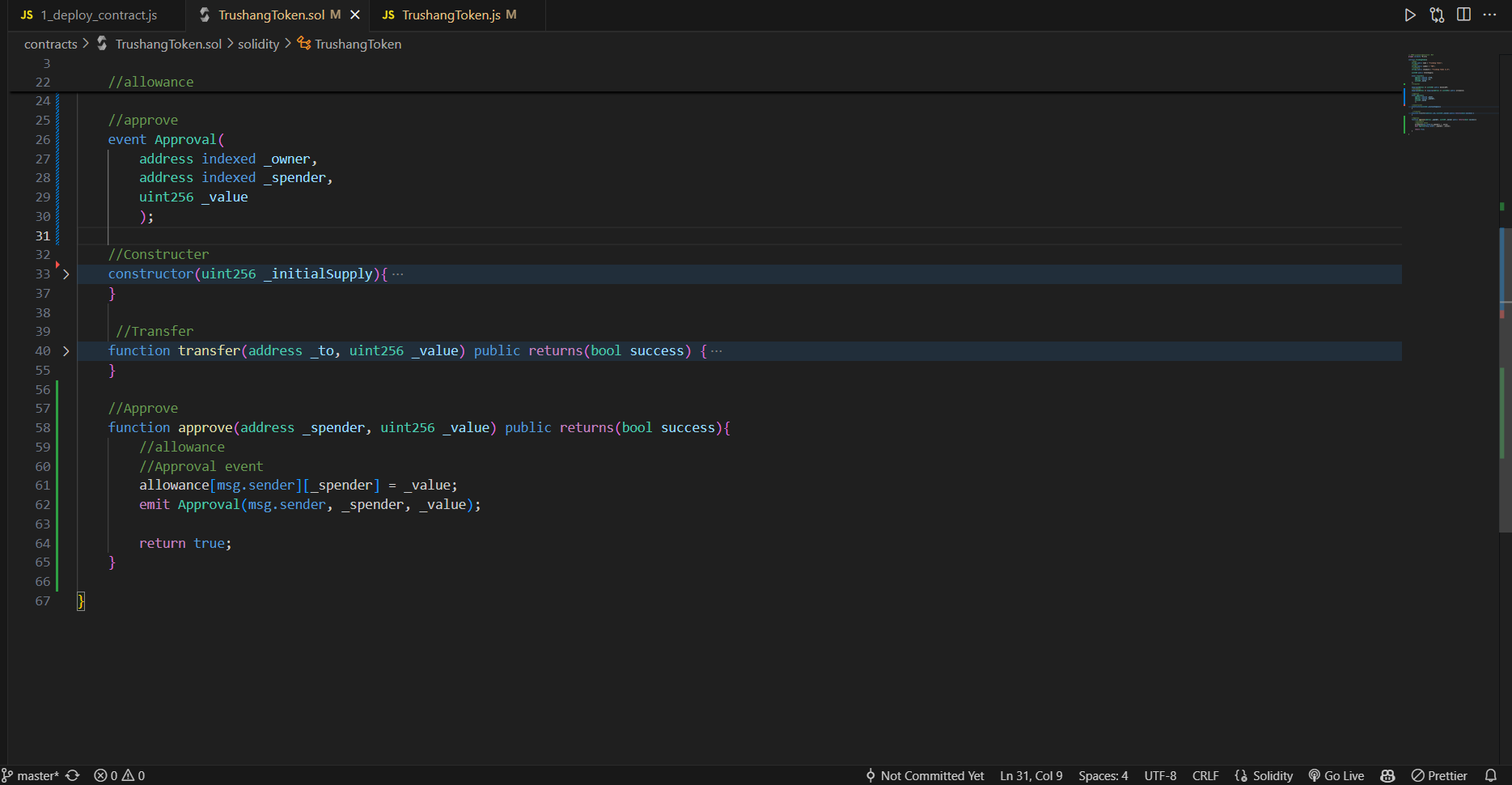


Figure 25:Create approve contract

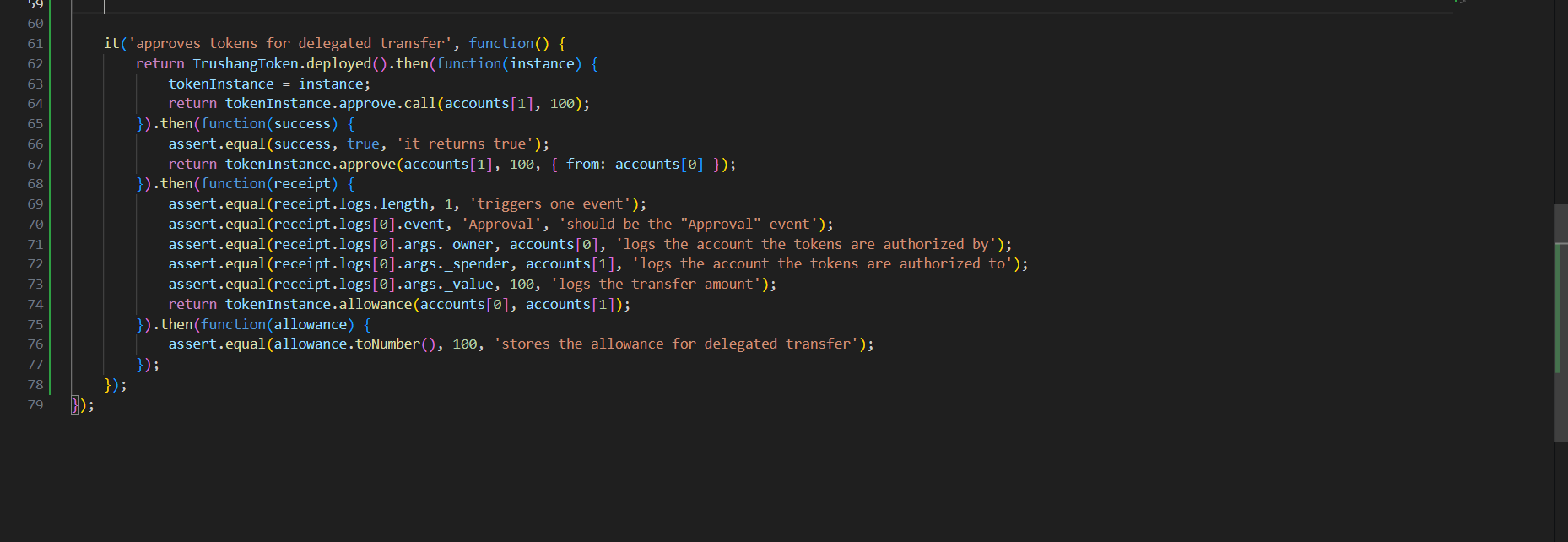


Figure 26:Write Approves tokens for delegated transfer function

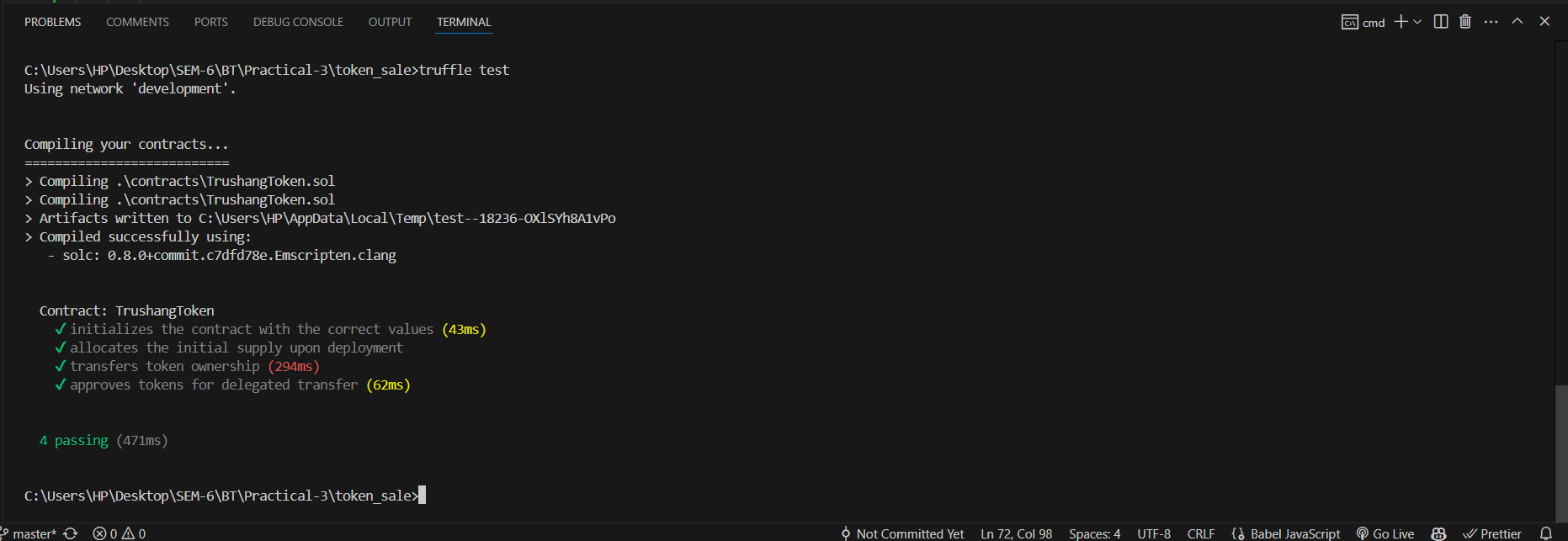


Figure 27:Test Approve token

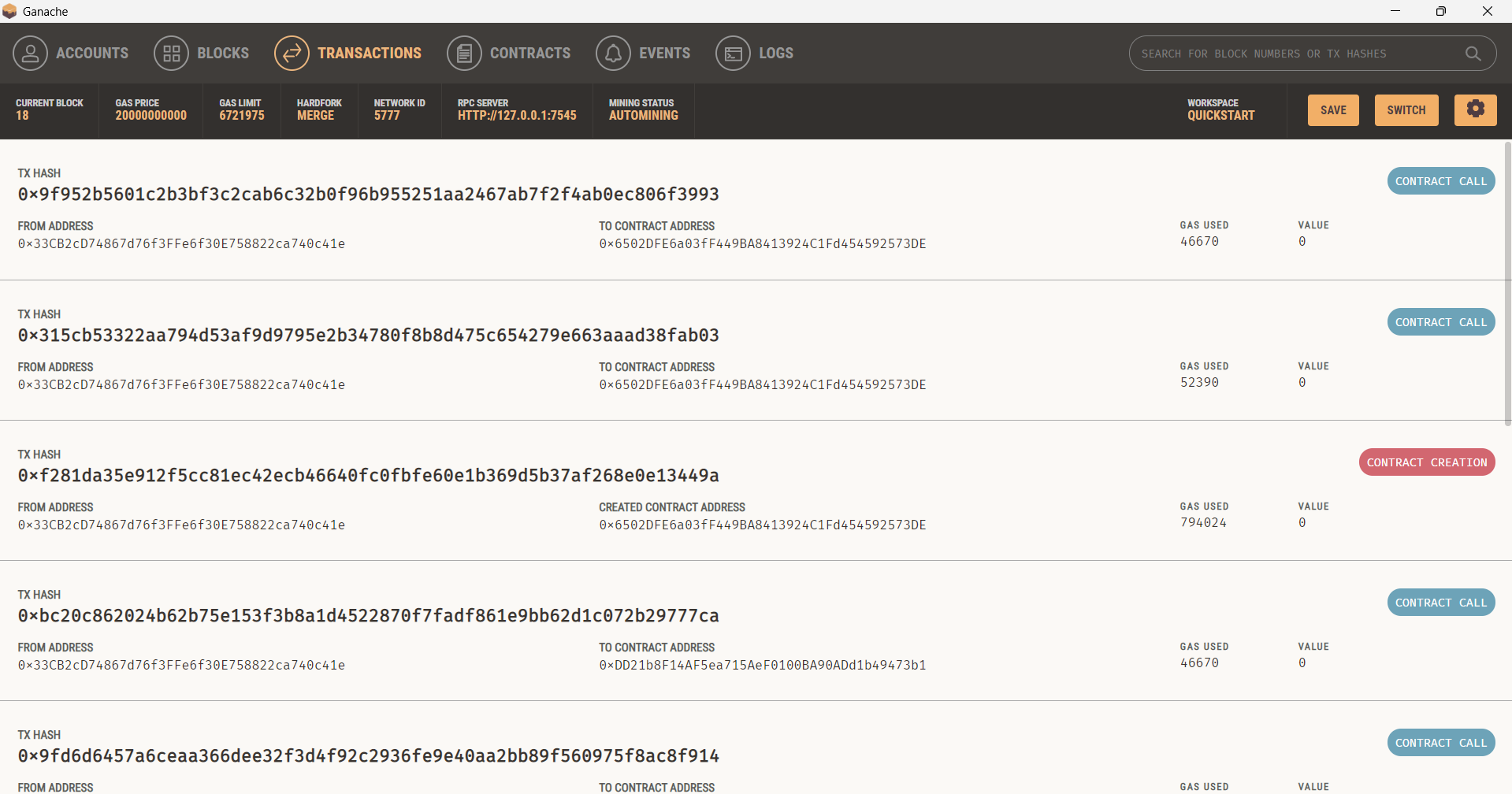


Figure 28:Approved token transfer in Ganache

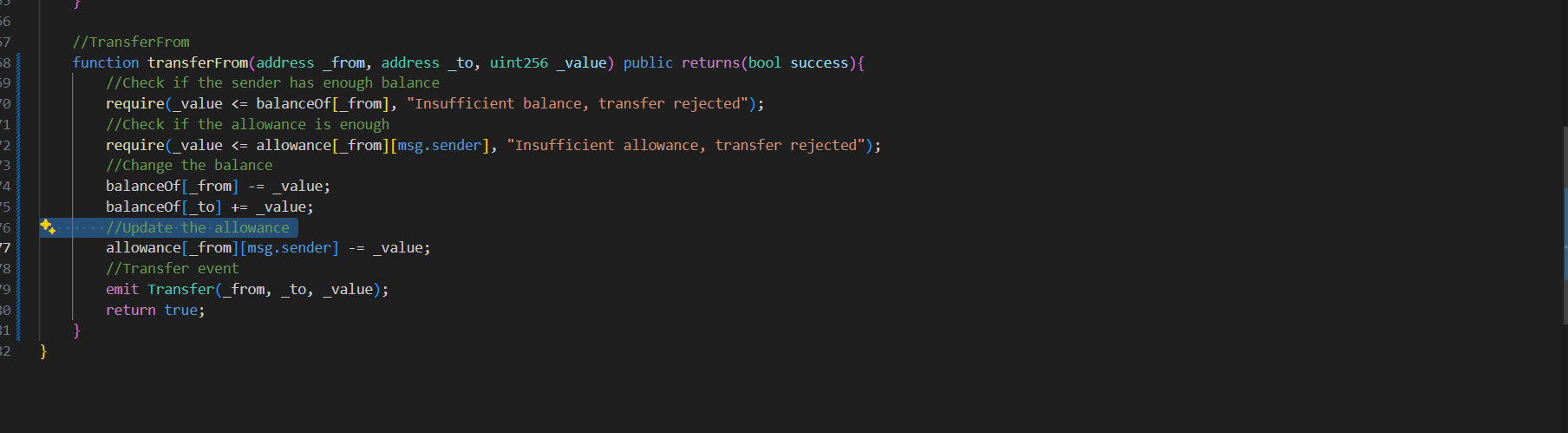


Figure 29:Write smart contract for transfer

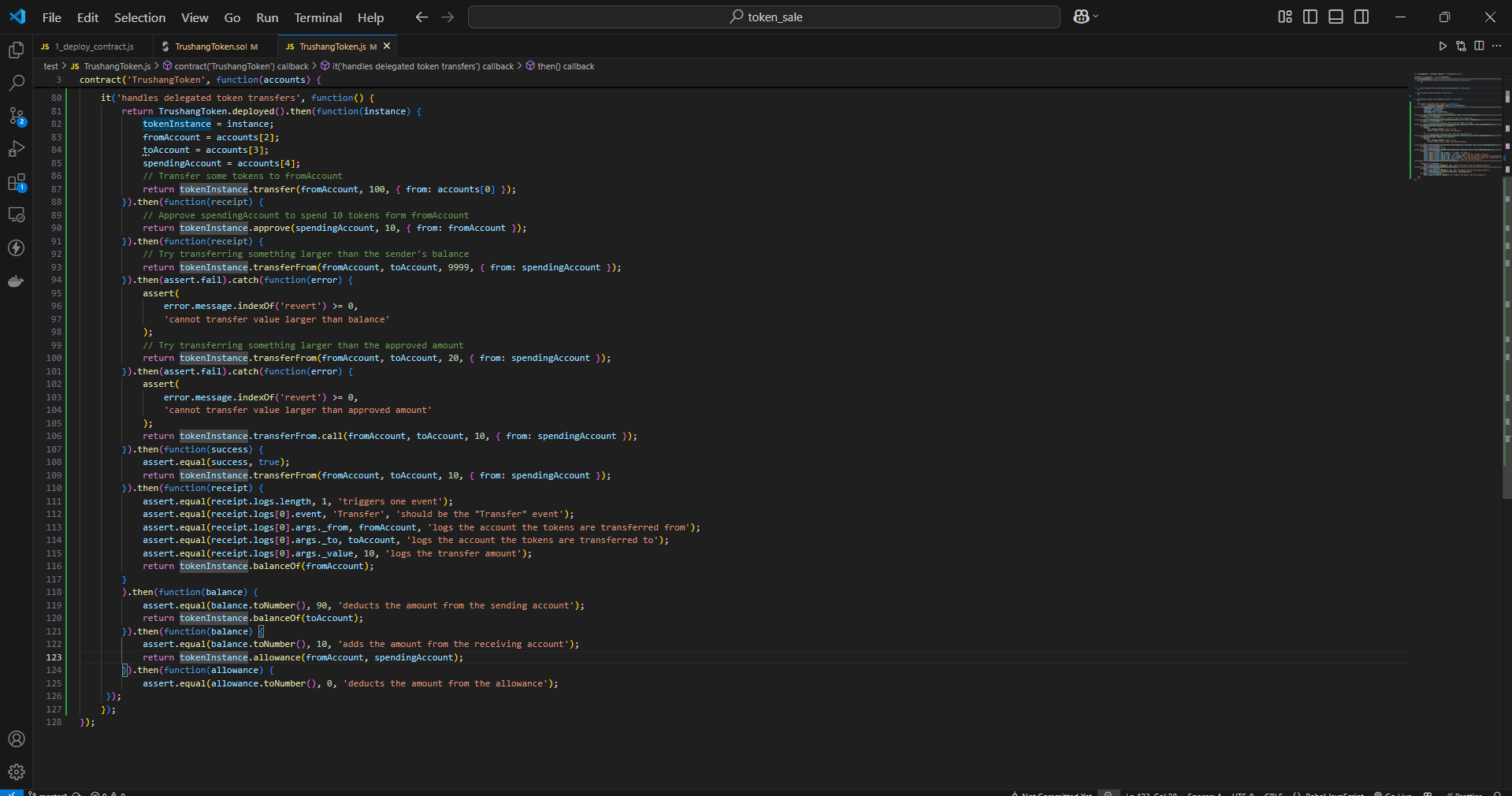


Figure 30:Transferfrom function

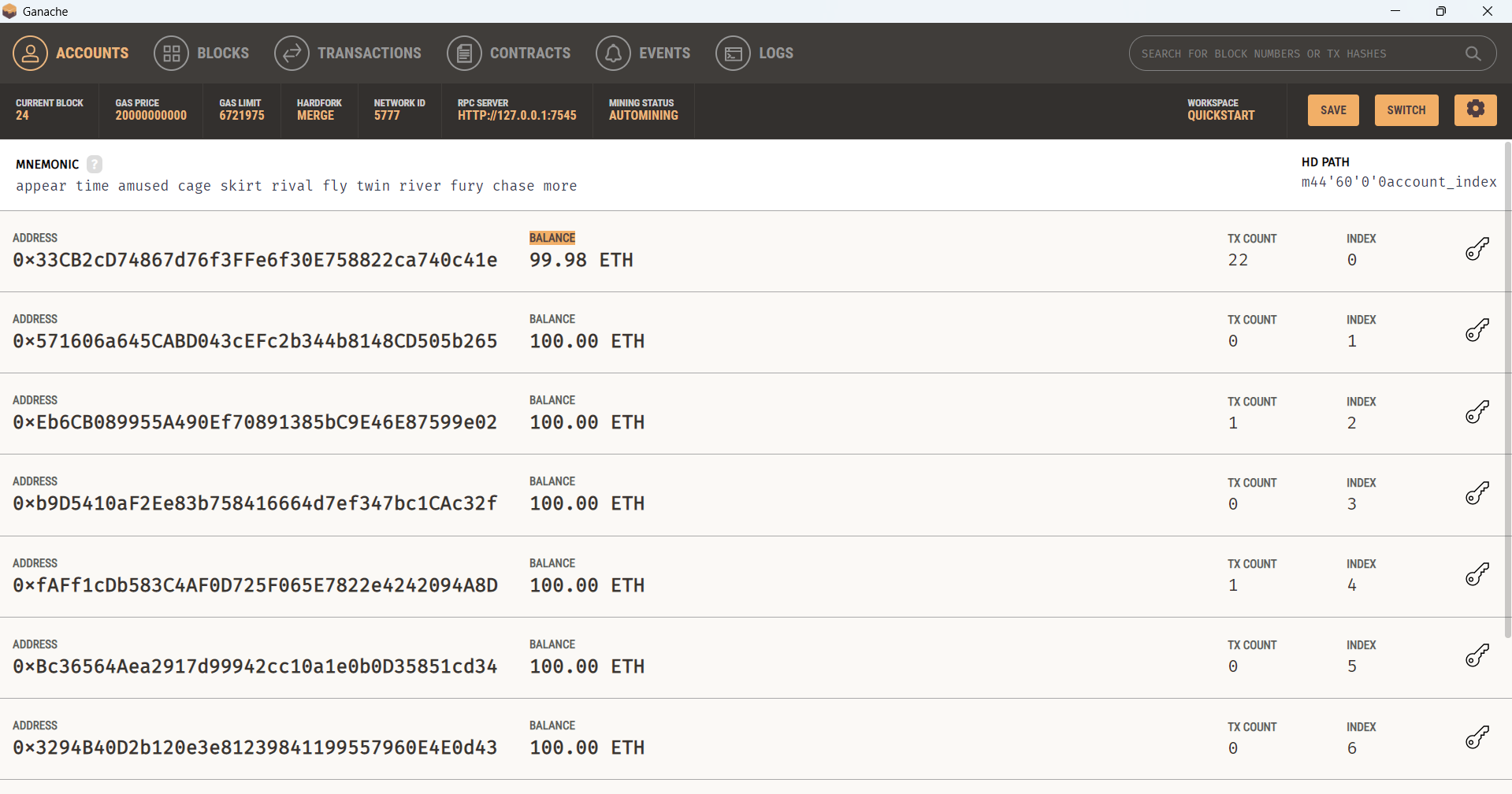


Figure 31:Ganache transaction

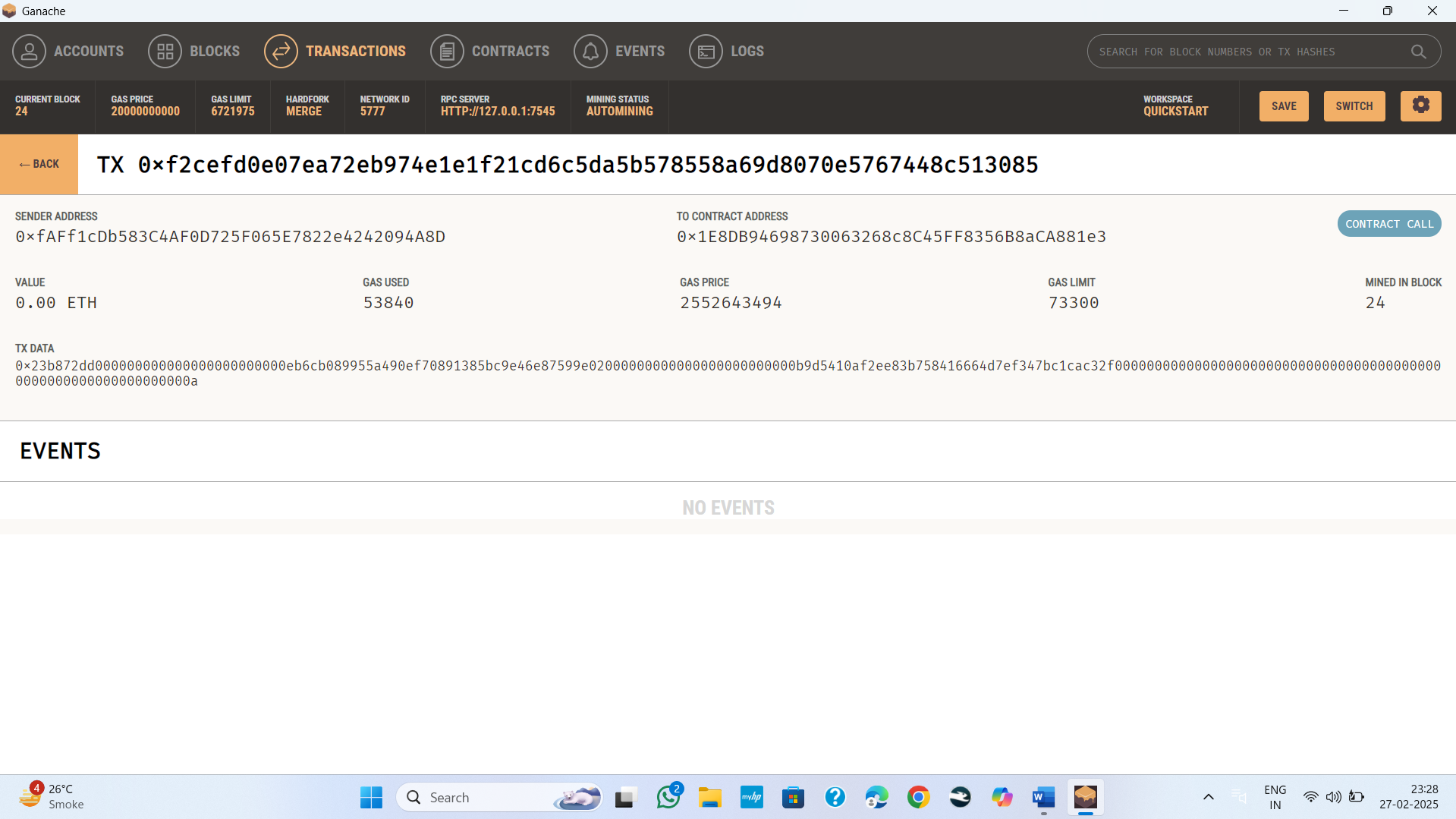


Figure 32:Single Ganache transaction

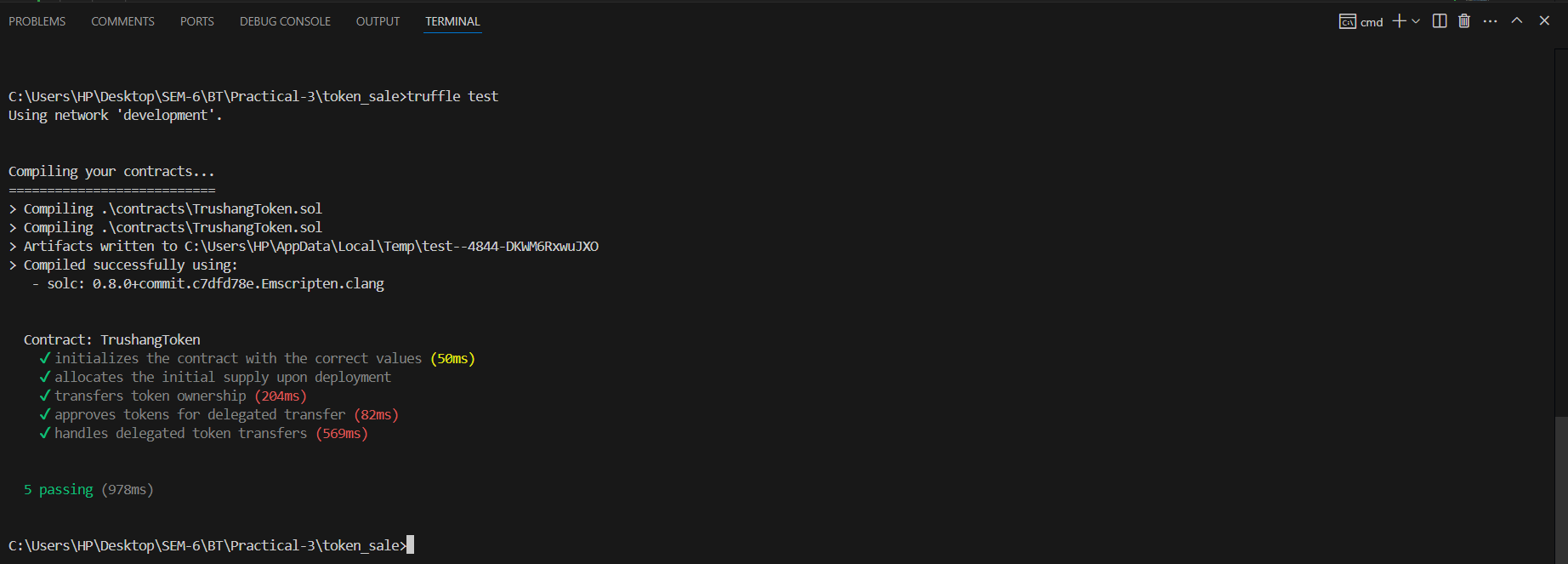


Figure 33:Test Delegated Token Transfers

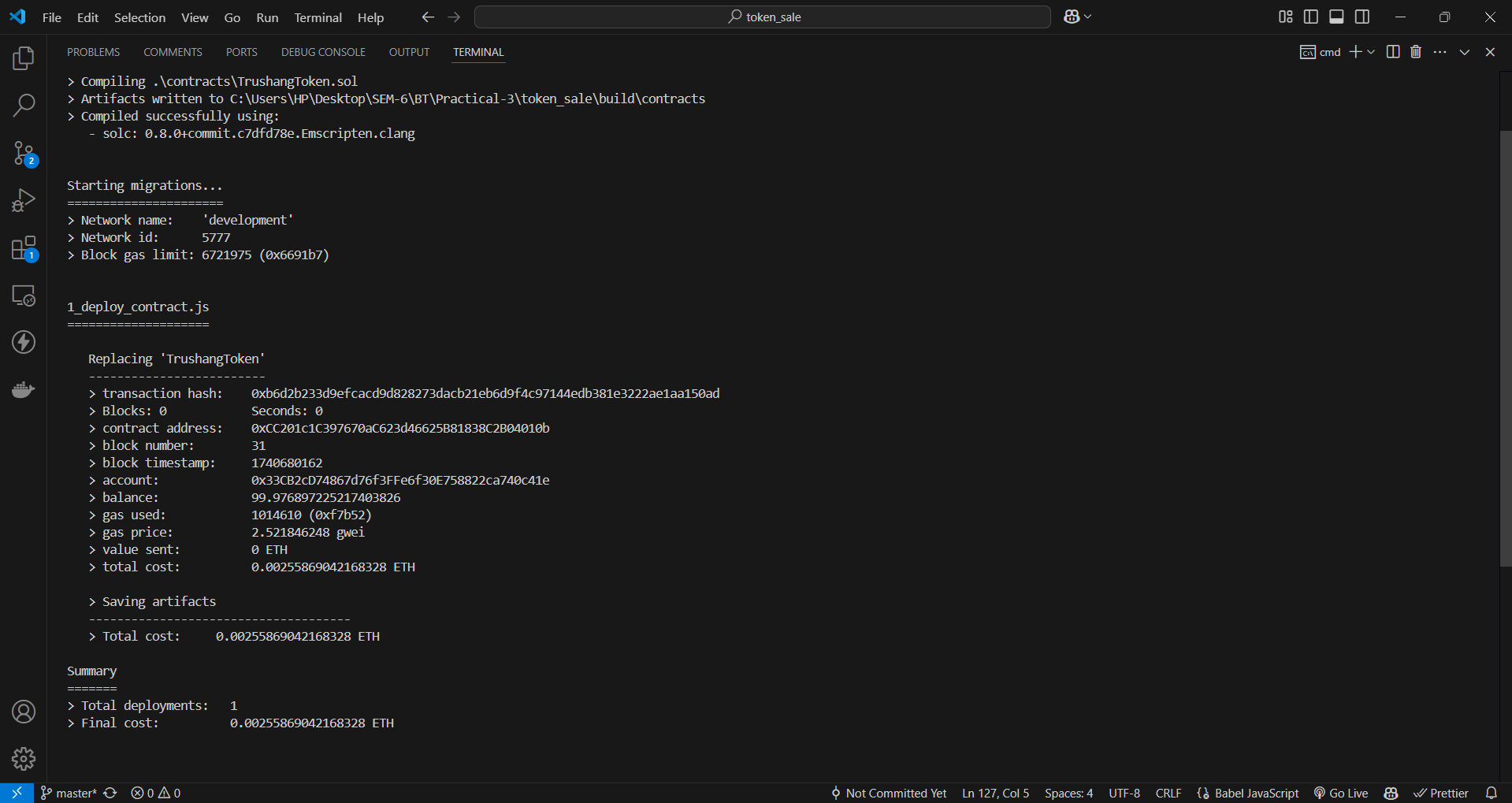


Figure 34:Migrate the contract

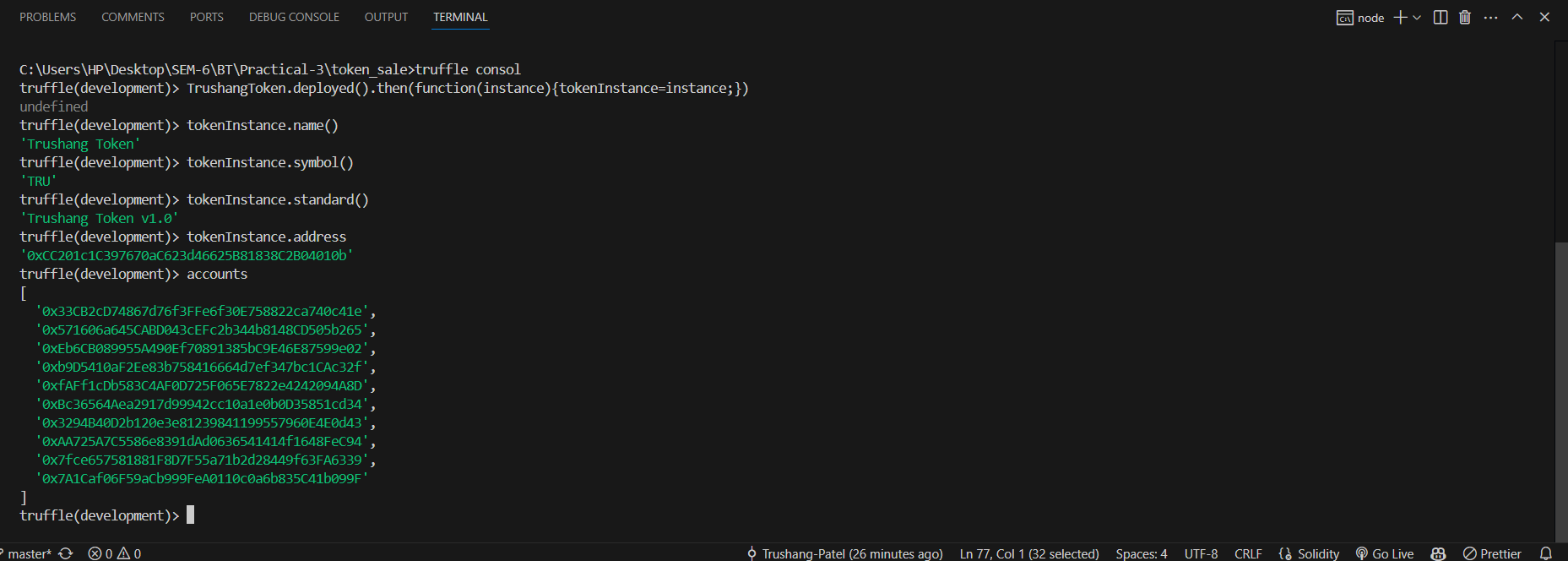


Figure 35:Check accounts in our Blockchain



Figure 36:Transfer token from adim to account[1]



Figure 37:See transaction Ganache

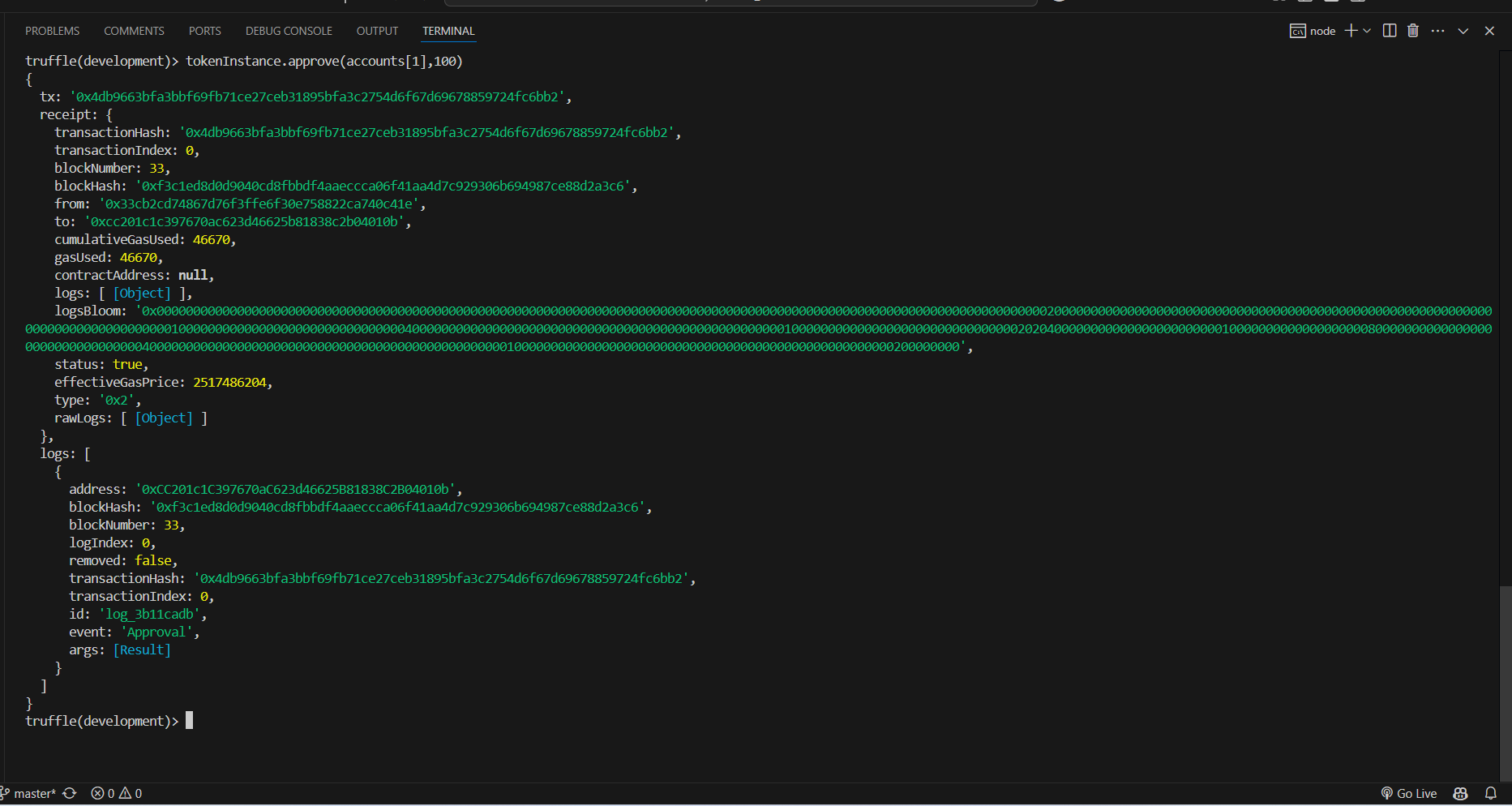


Figure 38:Approve transaction

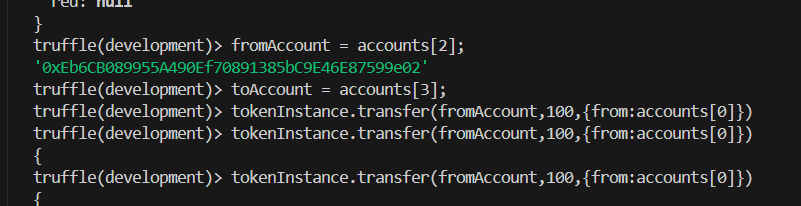


Figure 39:Transfer token

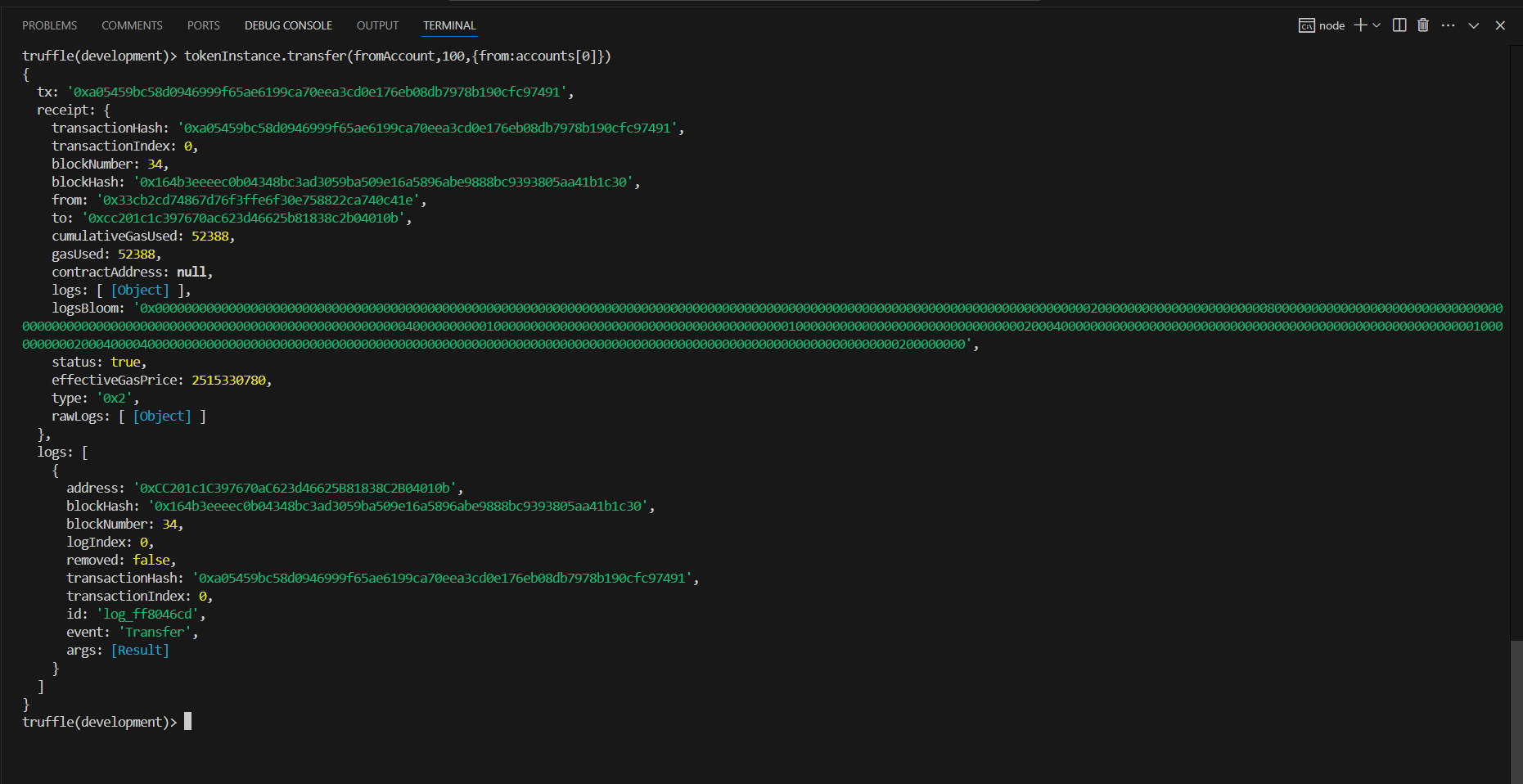


Figure 40:Transfer the token

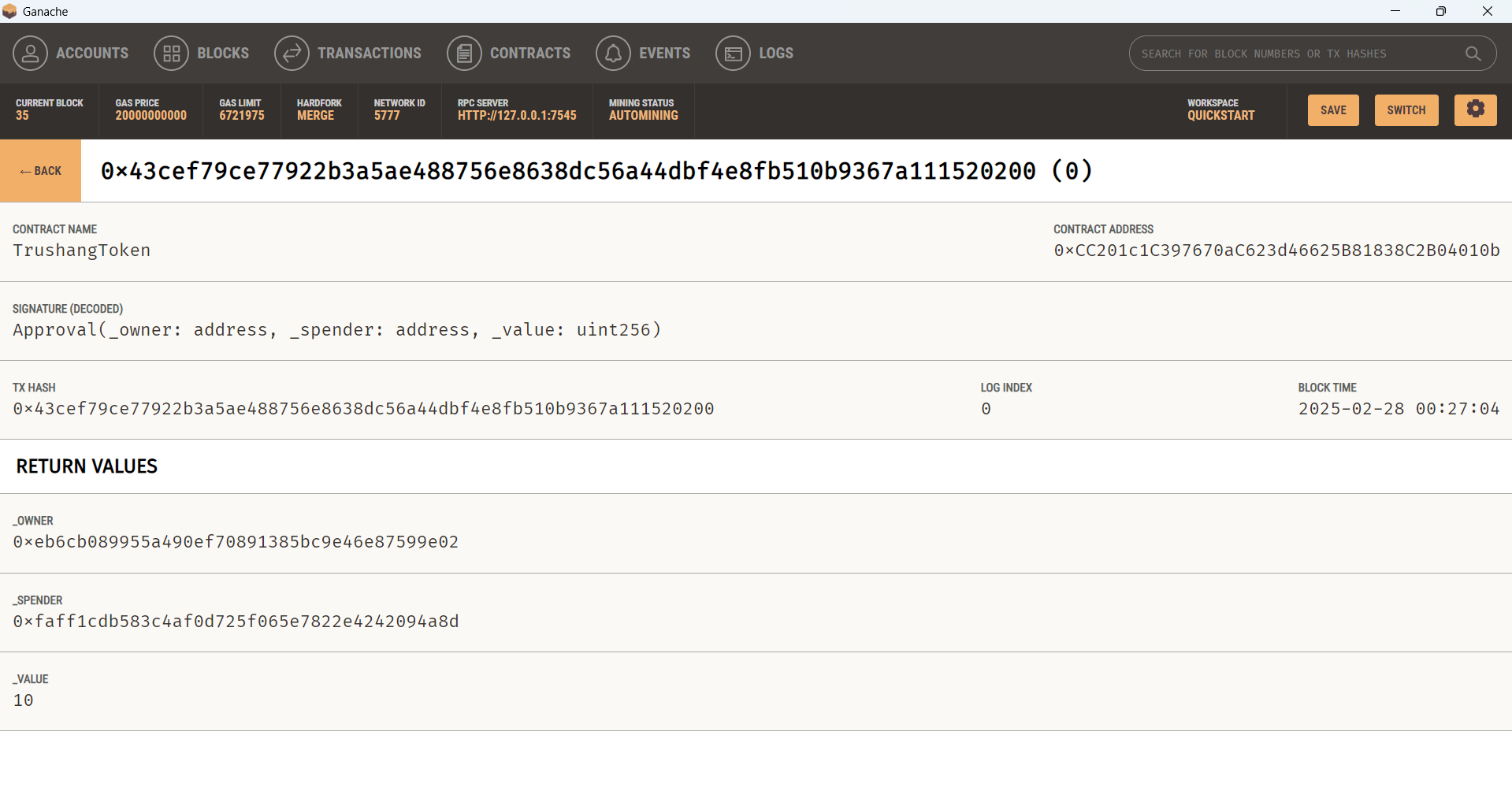


Figure 41:Transfer token transaction

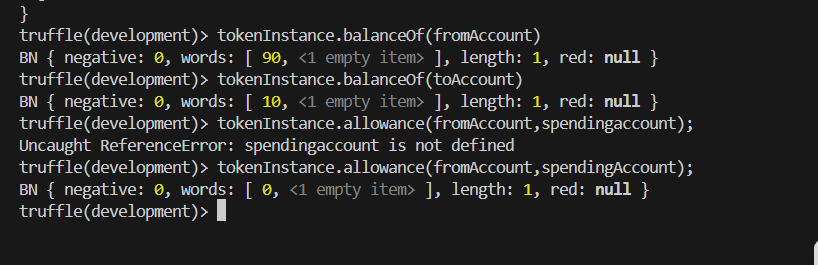


Figure 42:See the Balance of account

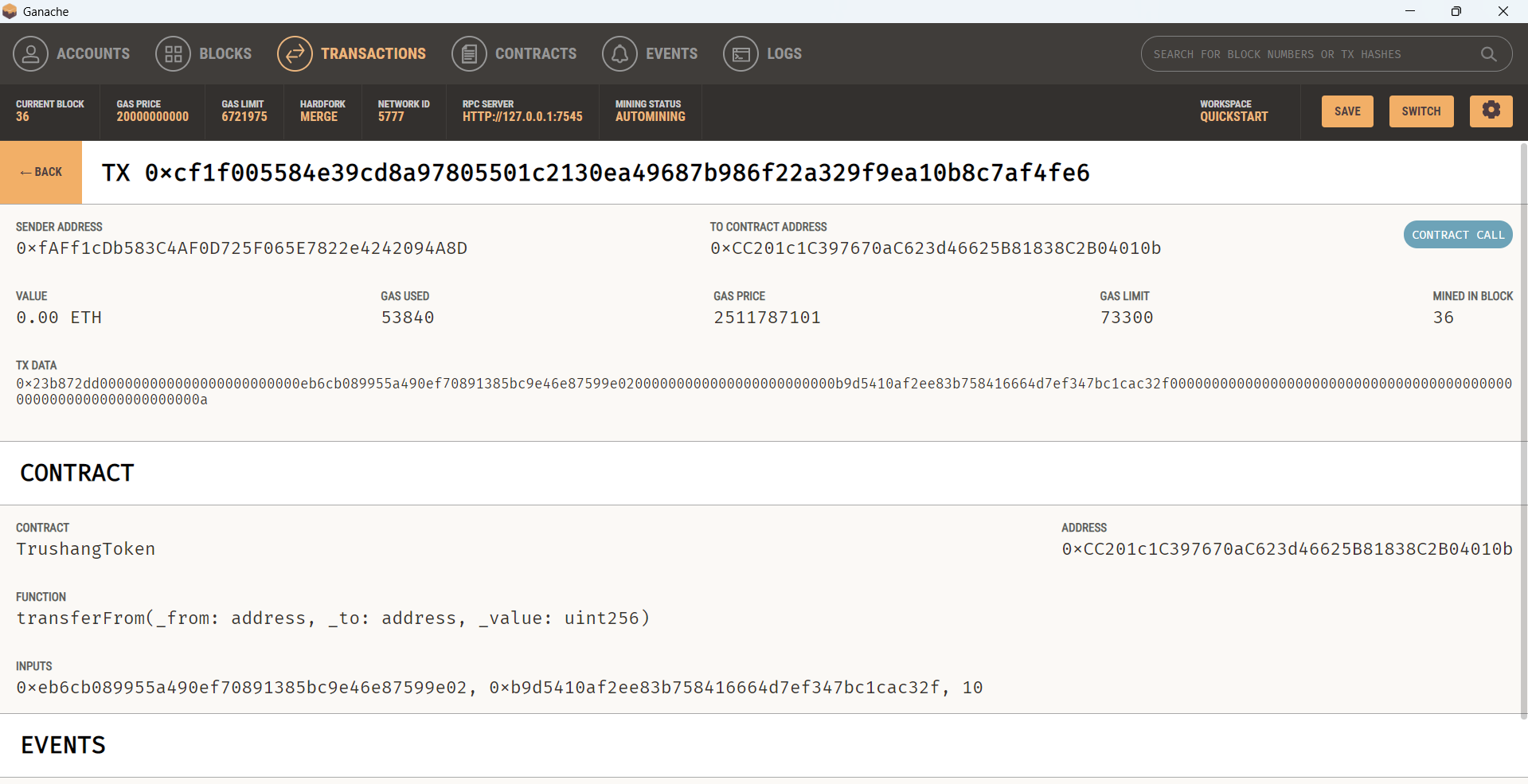


Figure 43:Transferform function transaction

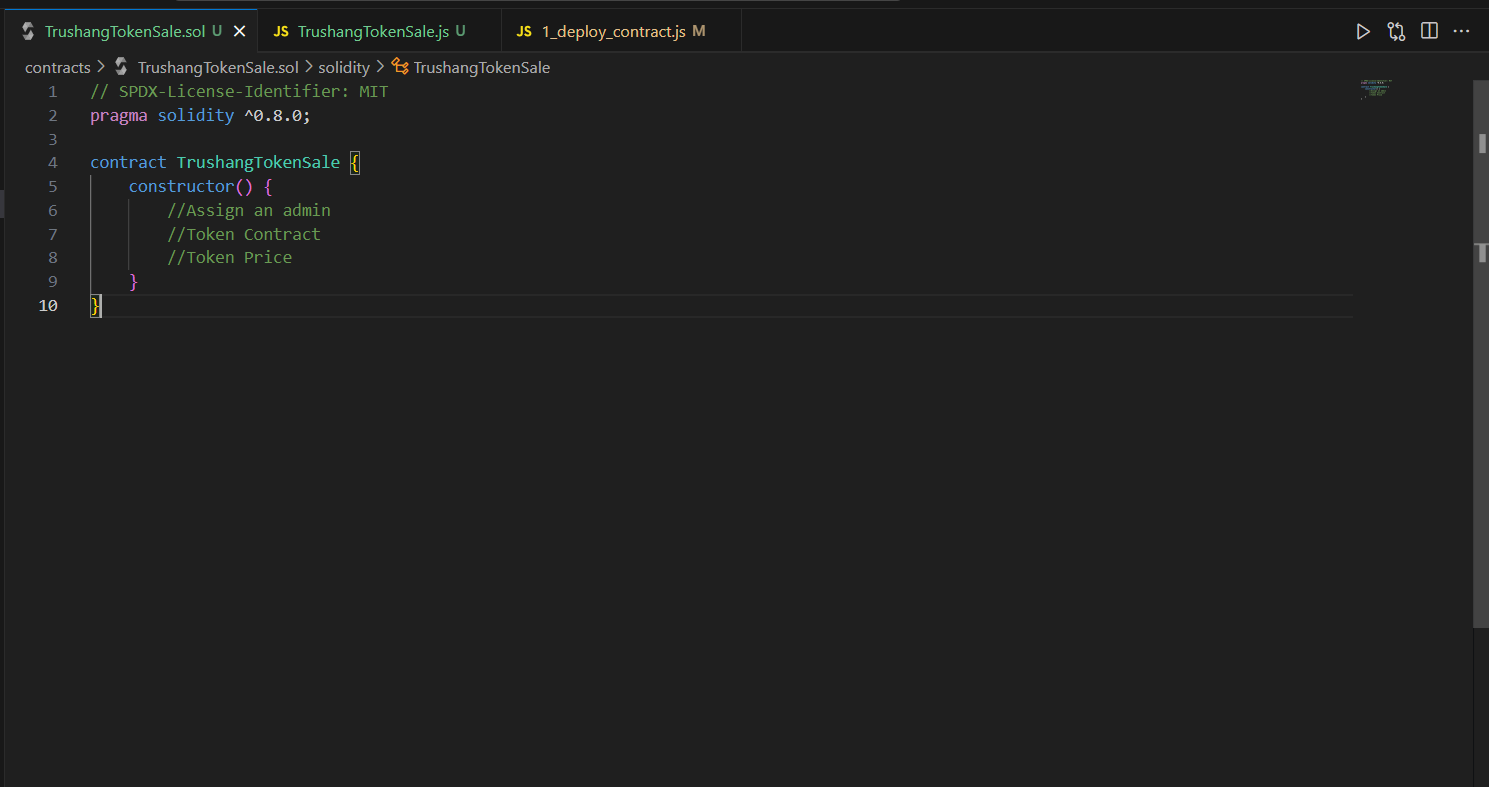


Figure 44:Start Tokensale contract

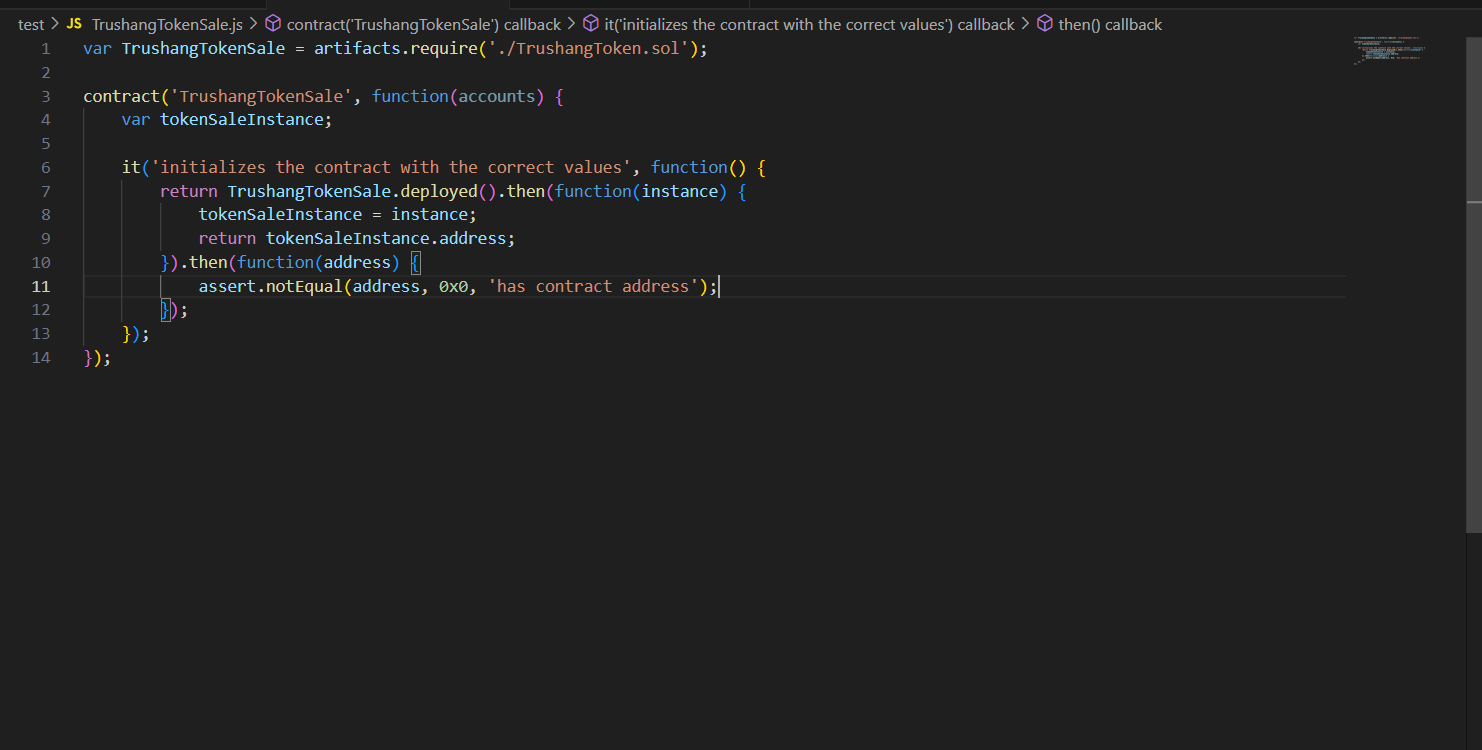


Figure 45:Write Tokensale contract

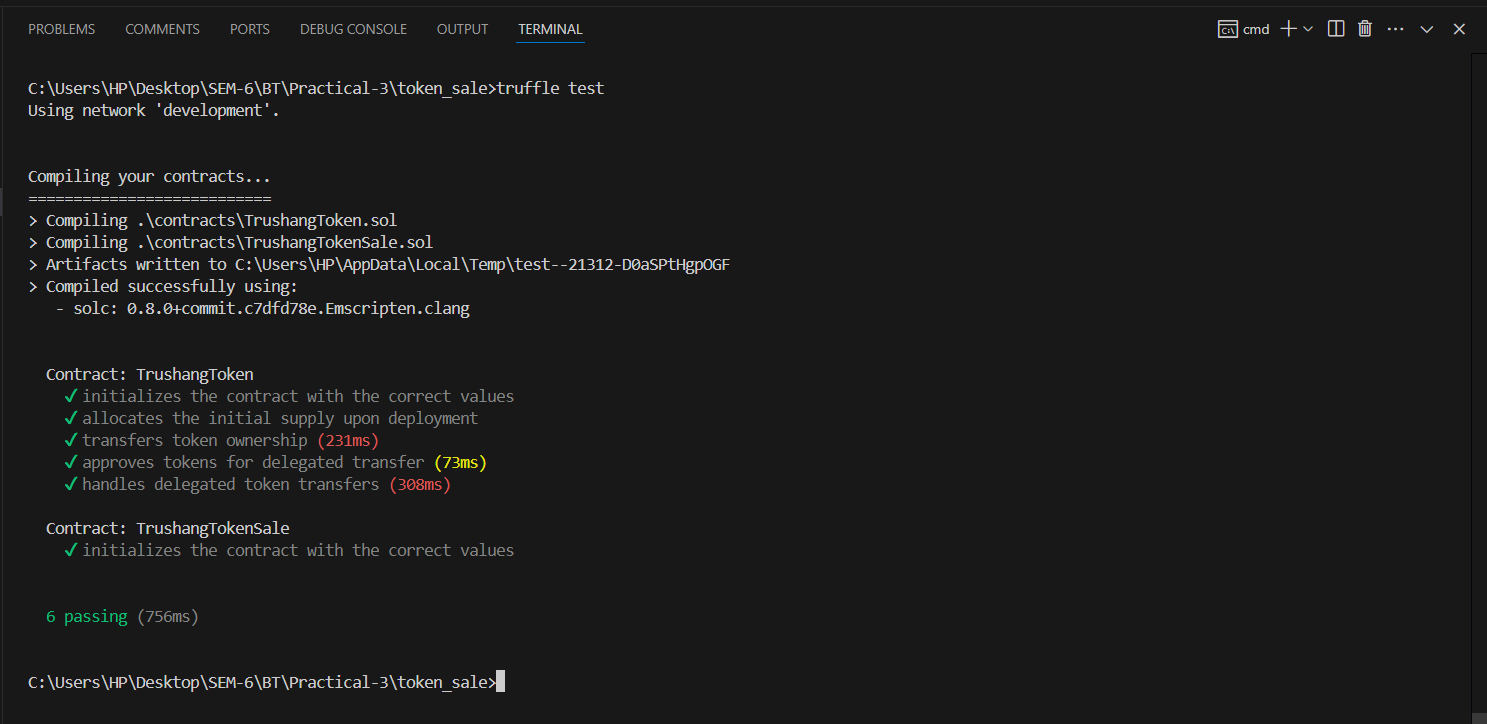


Figure 46:Test contract



Figure 47:Add tokensale in deployment contract

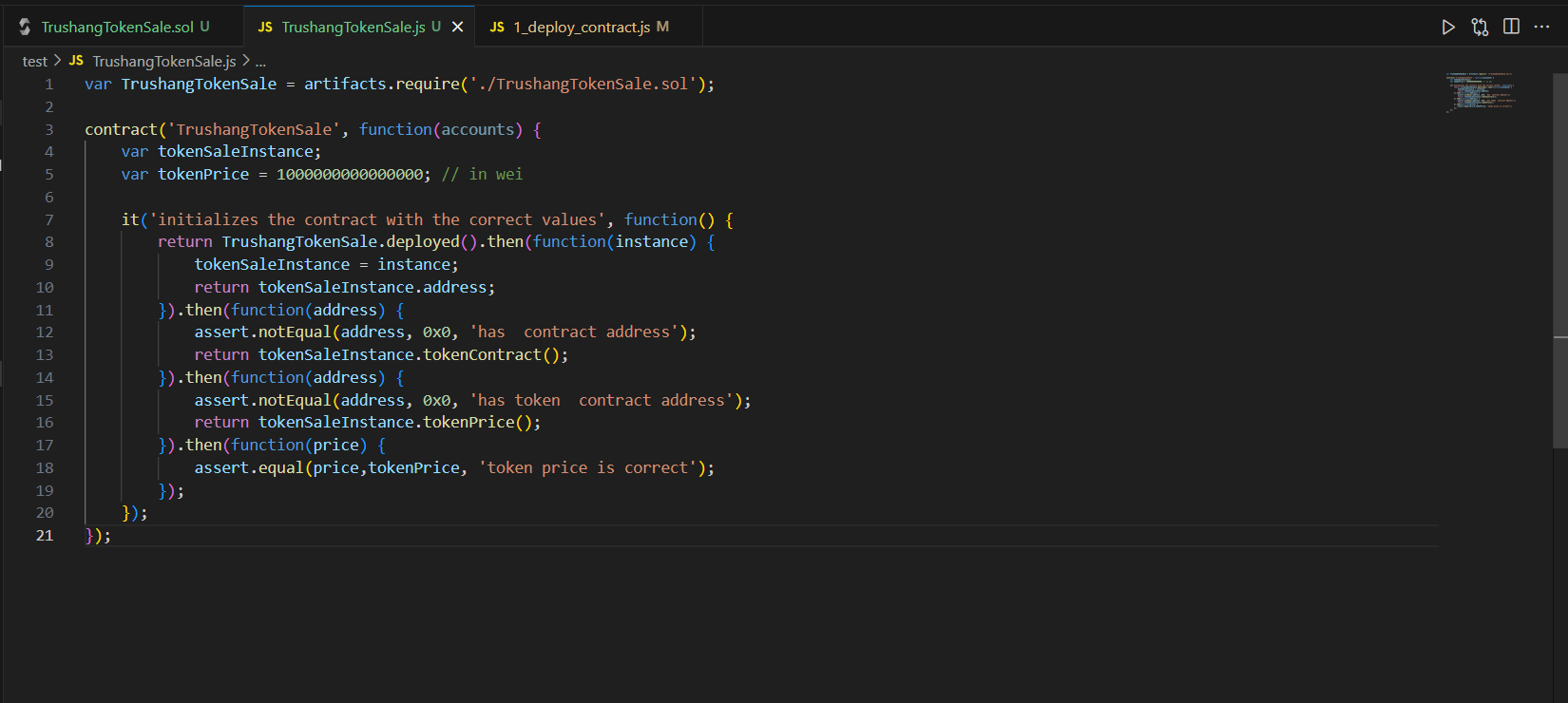


Figure 48:Write tokensale function

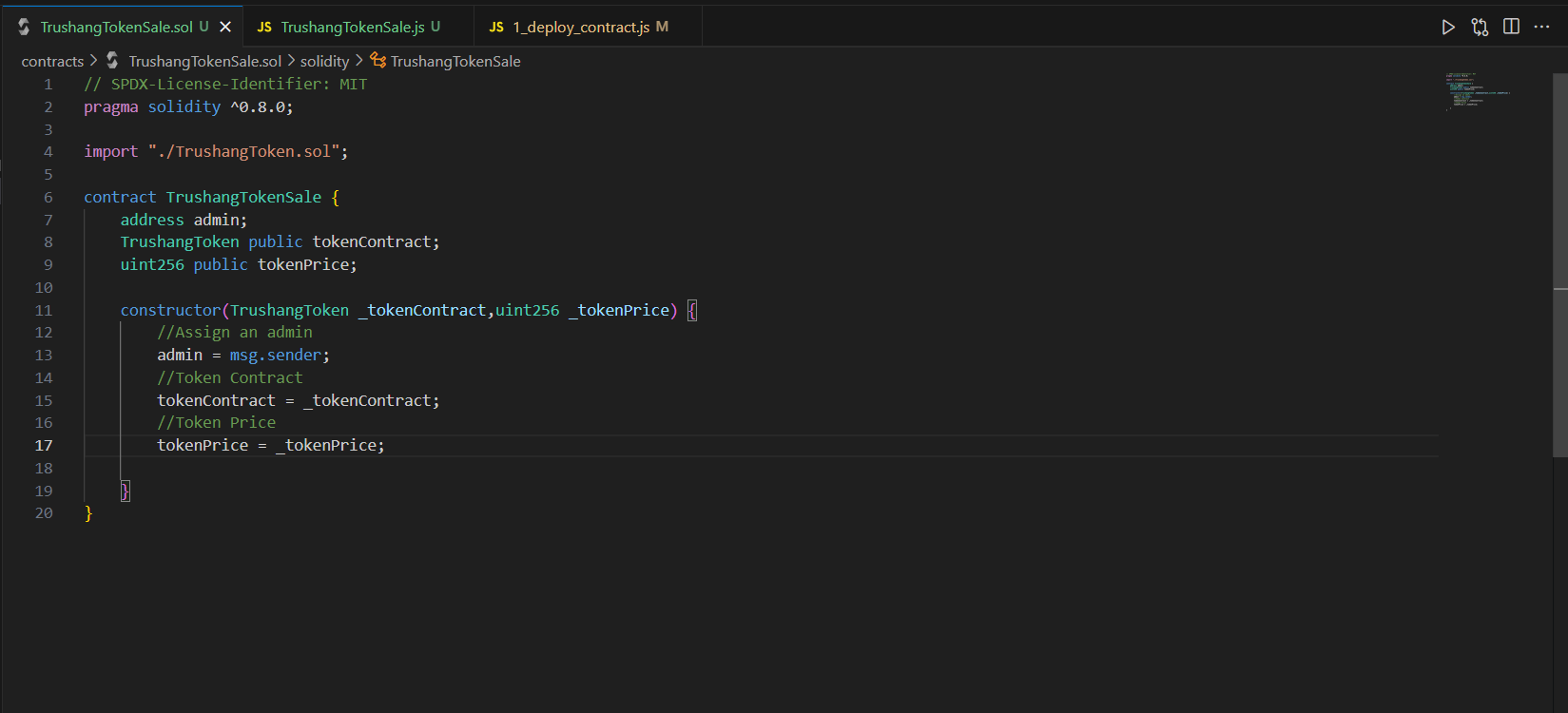


Figure 49: Write tokensale contract

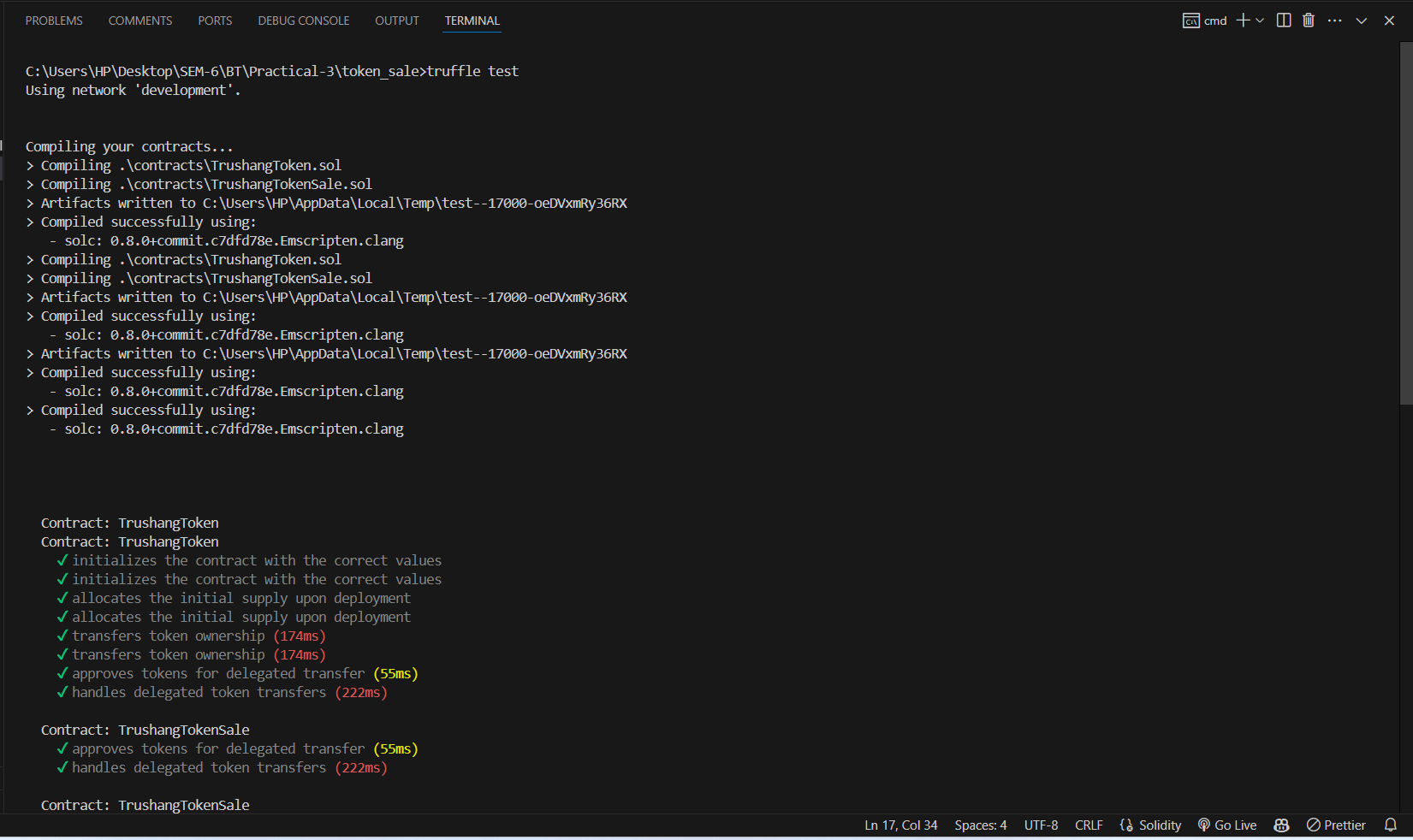


Figure 50:Test the contract

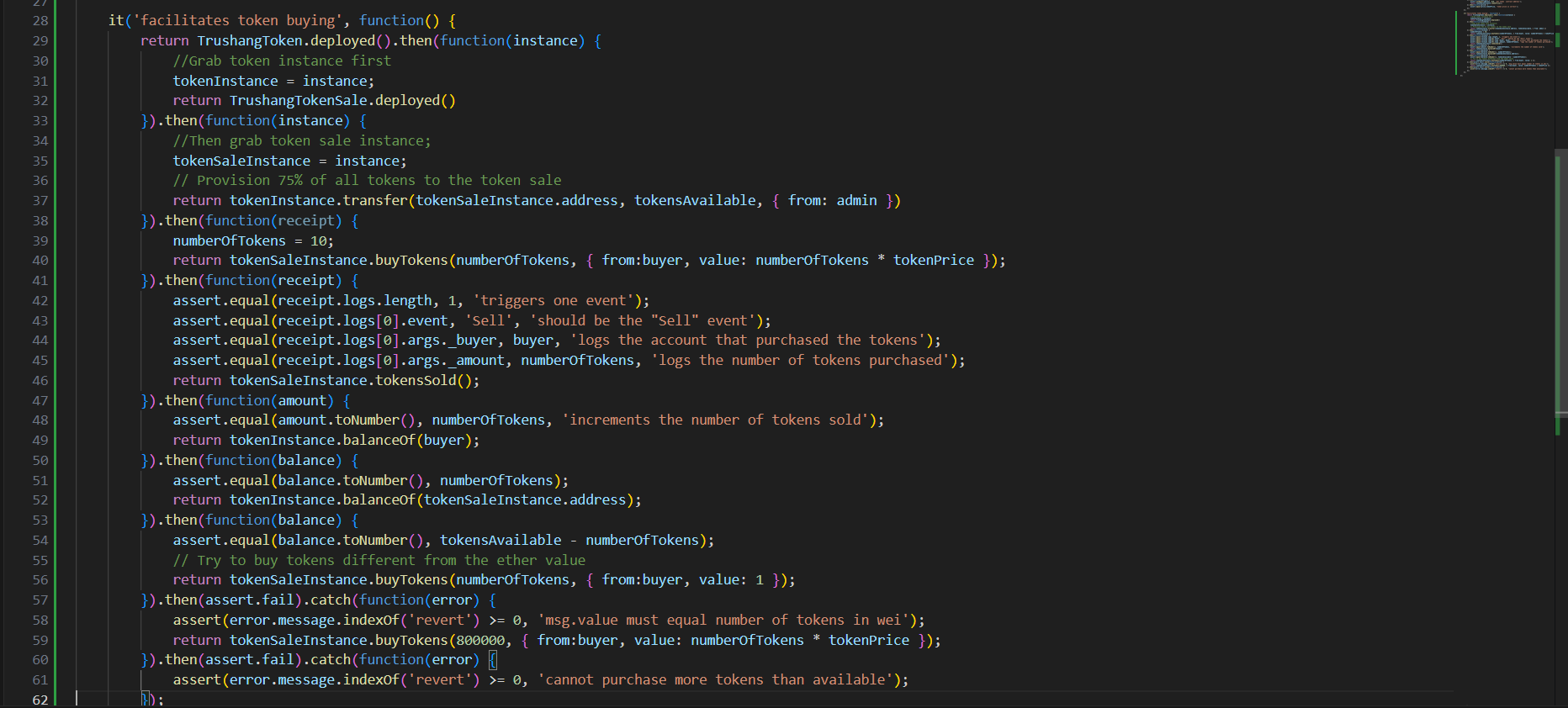


Figure 51:Write token buying function

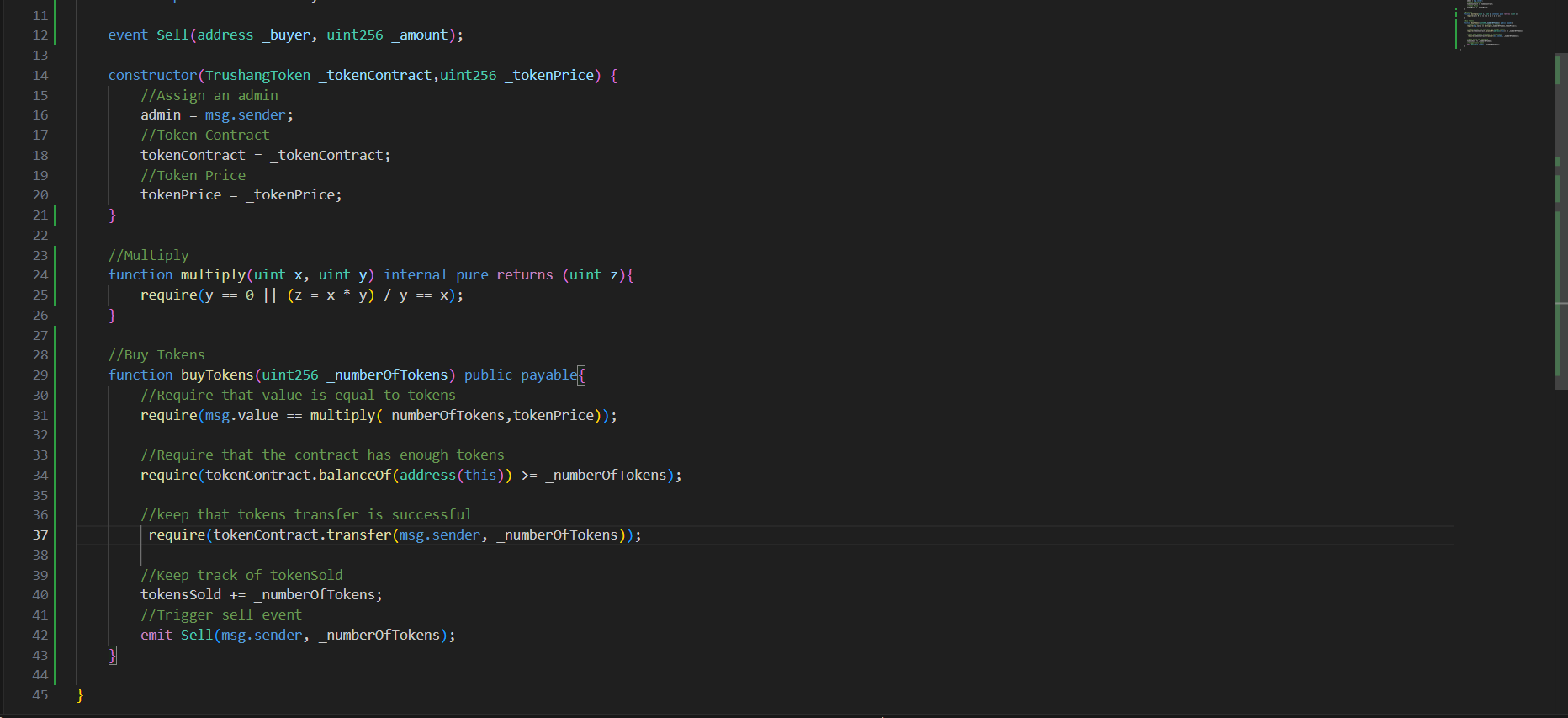


Figure 52:Write buytoken in contract

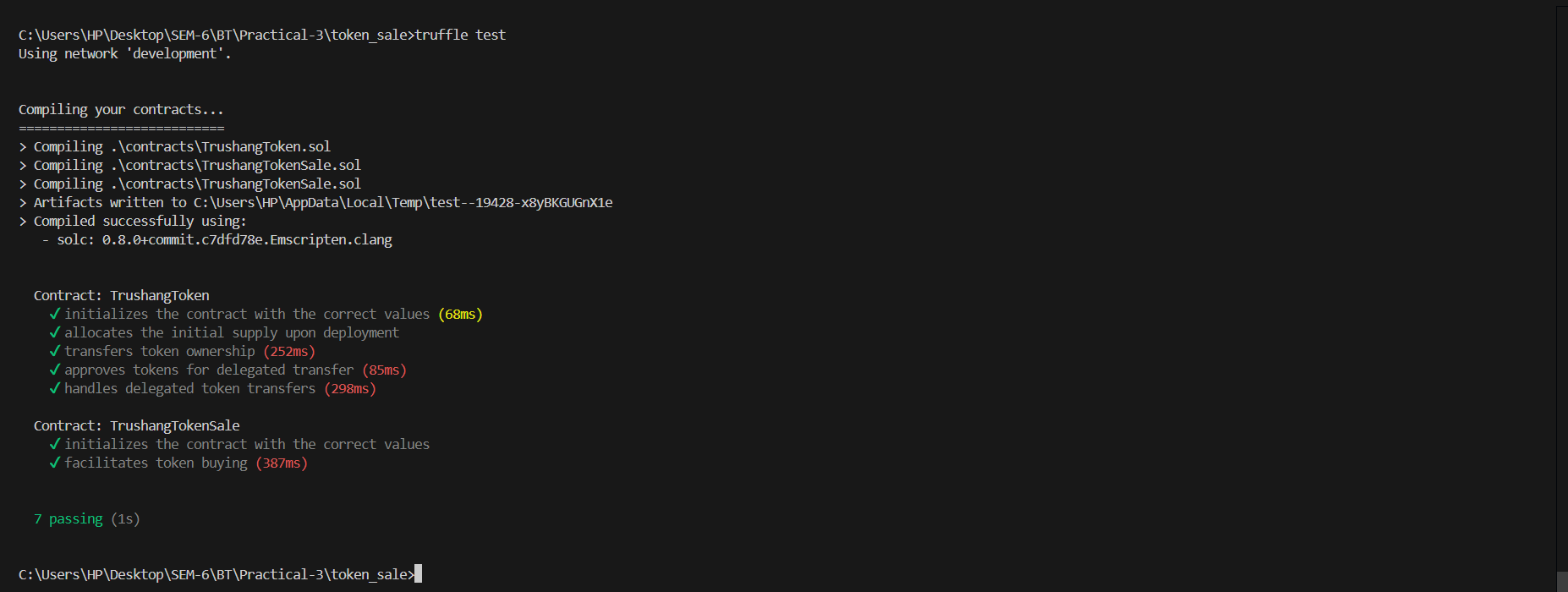


Figure 53:Deploy the transaction

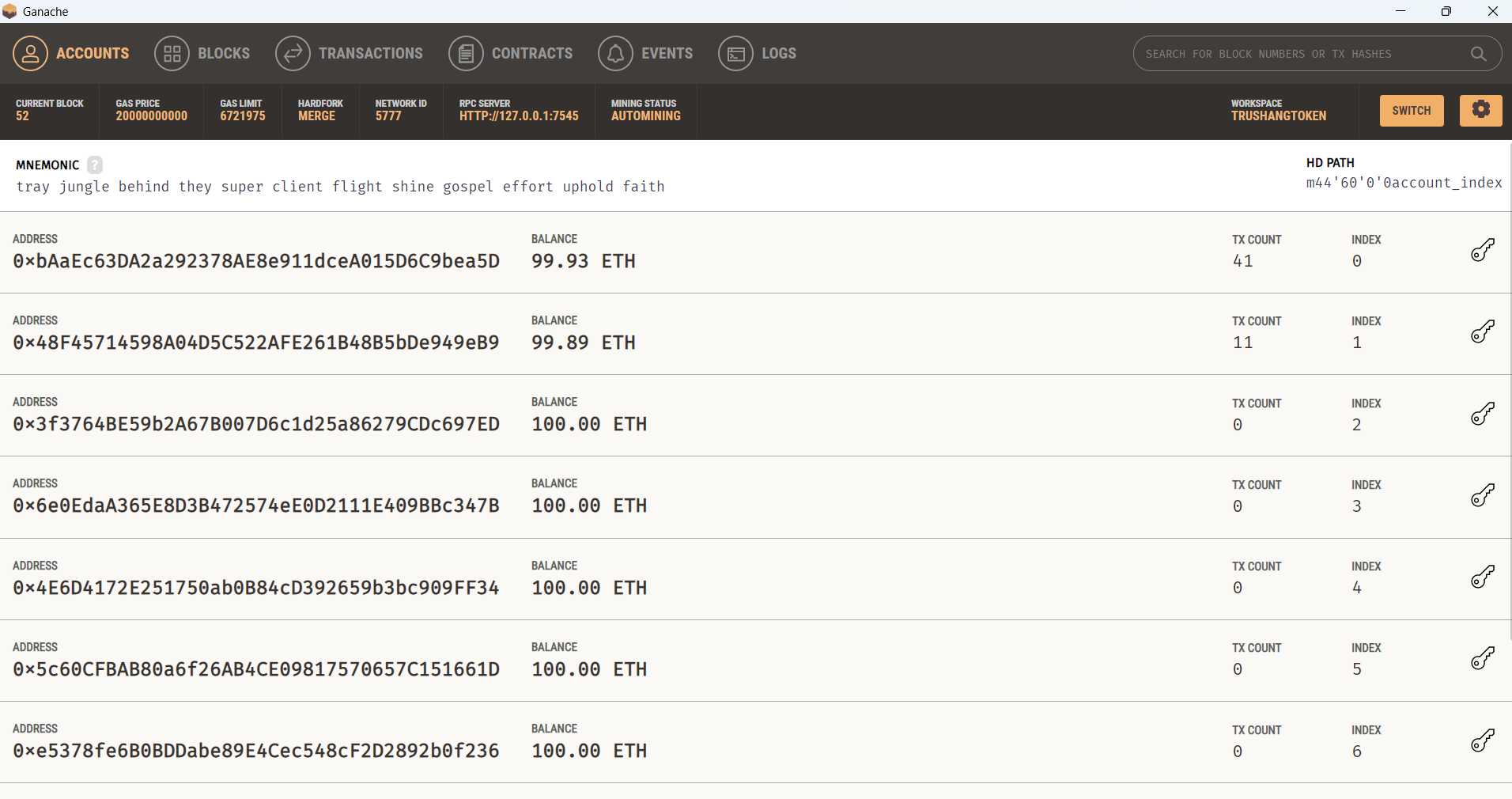


Figure 54:Ganache transaction

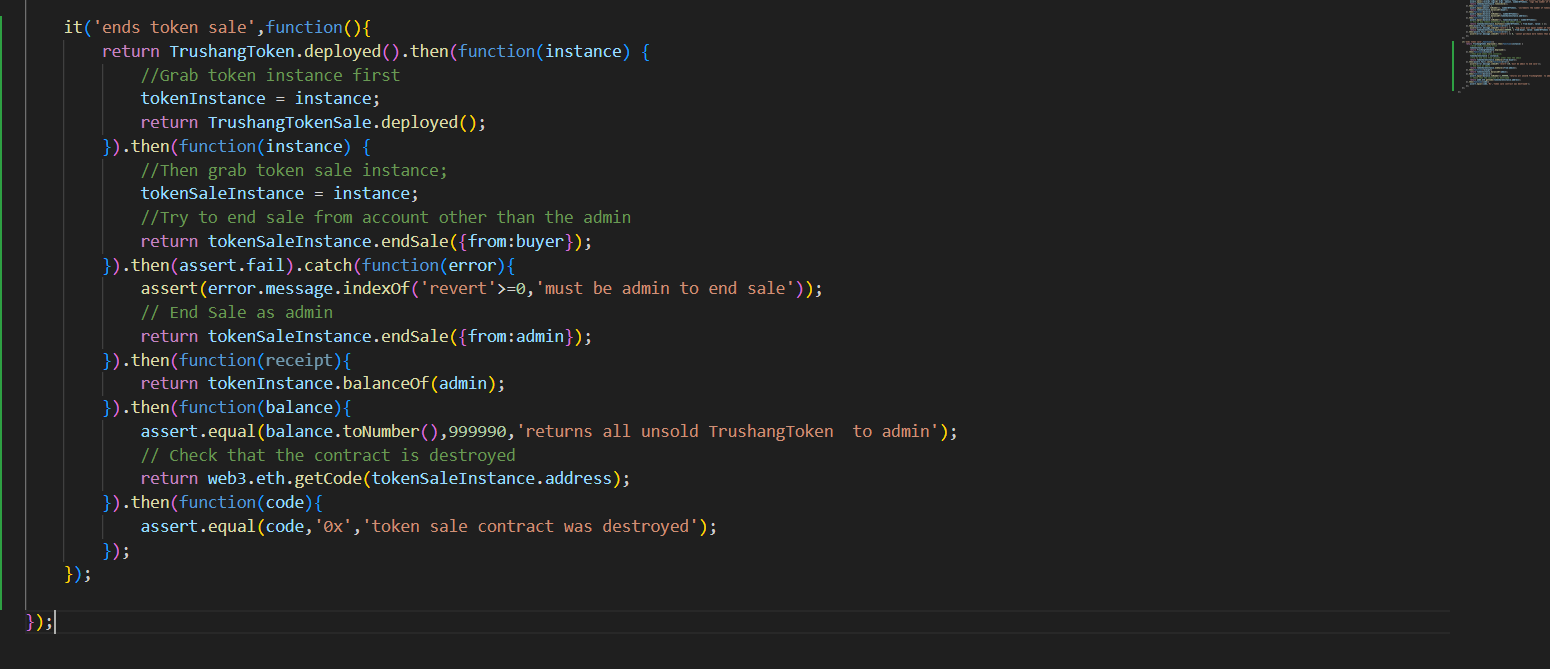


Figure 55:Ending Sale token function

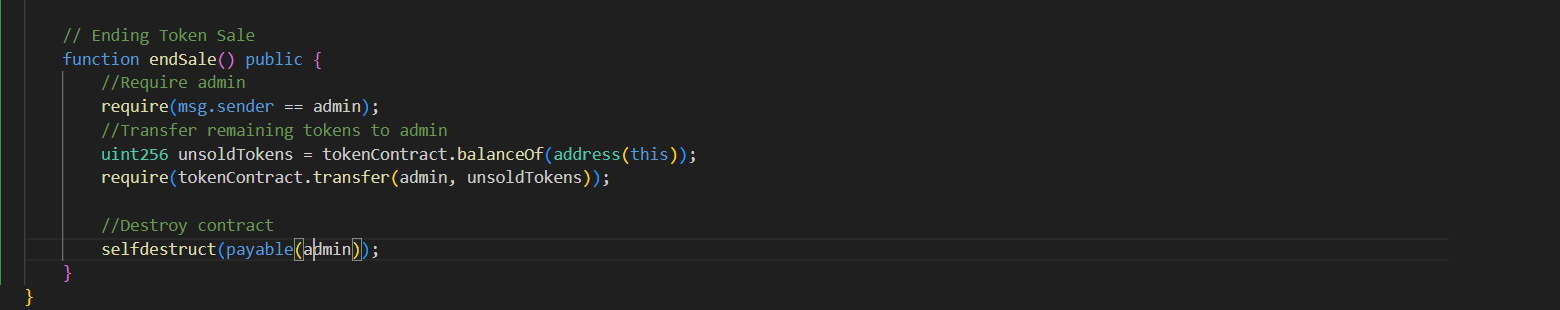


Figure 56:Ending Sale token in contract

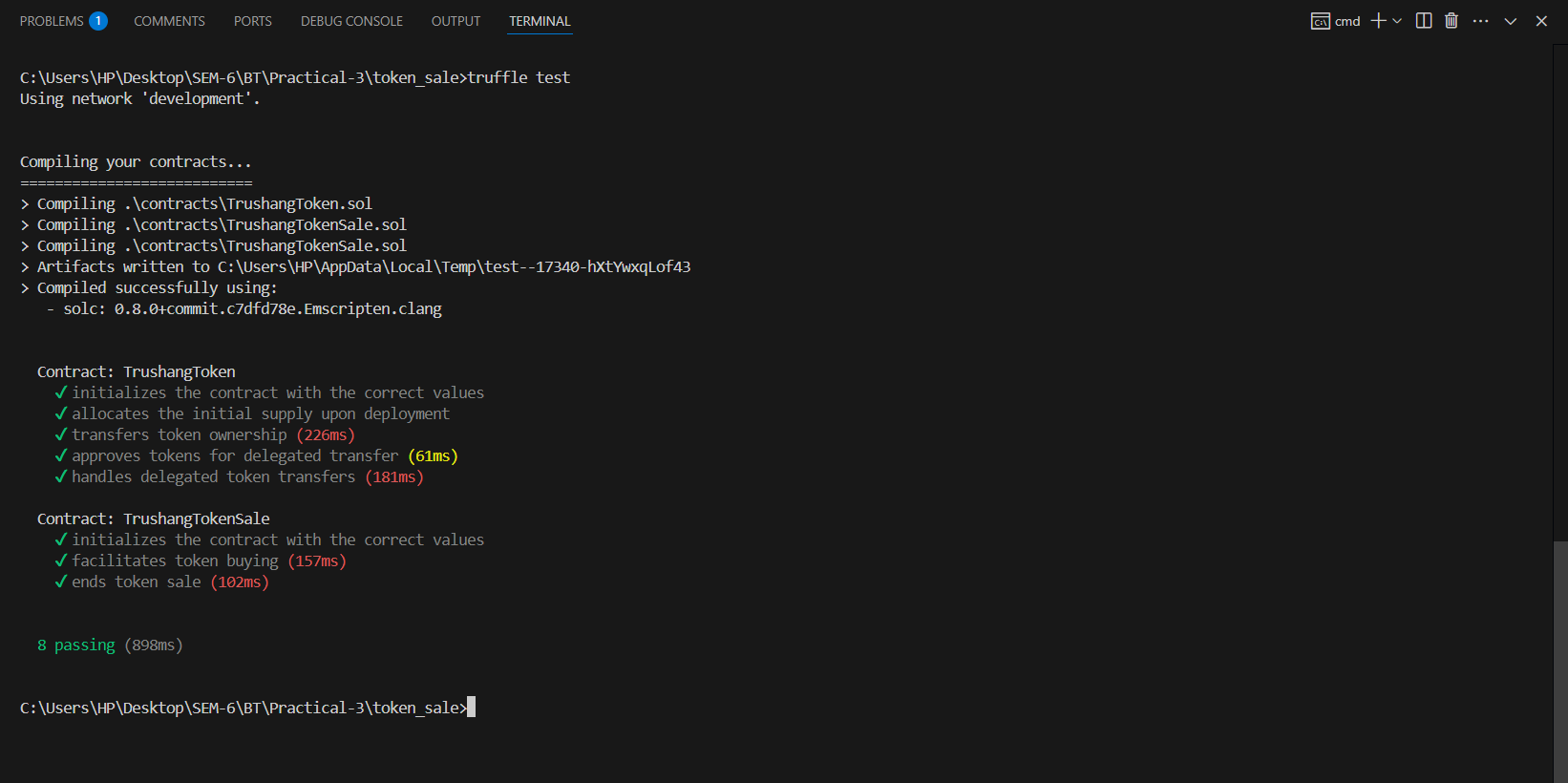


Figure 57:Deployed contract after end token sale

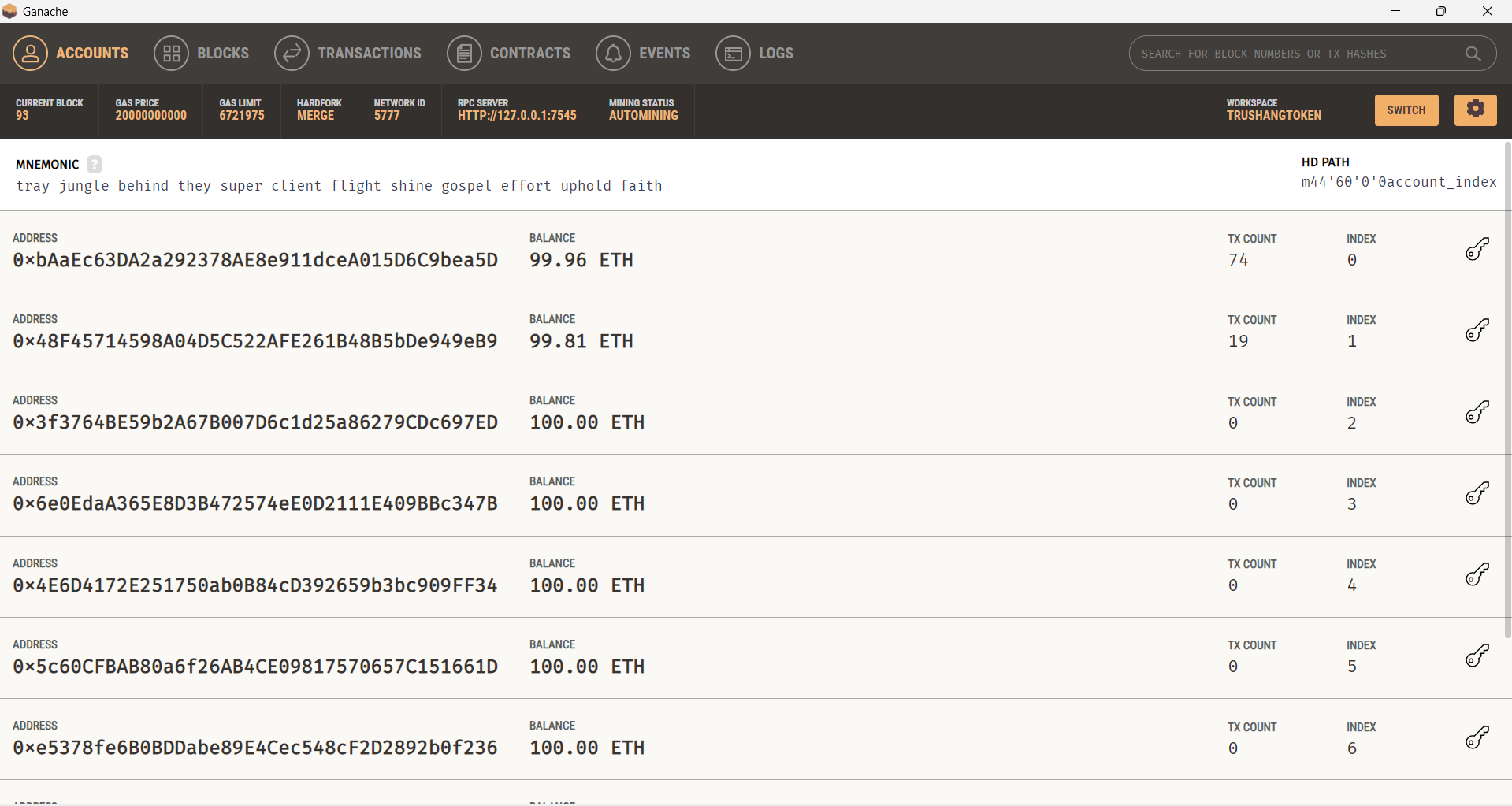


Figure 58:Ganache transaction

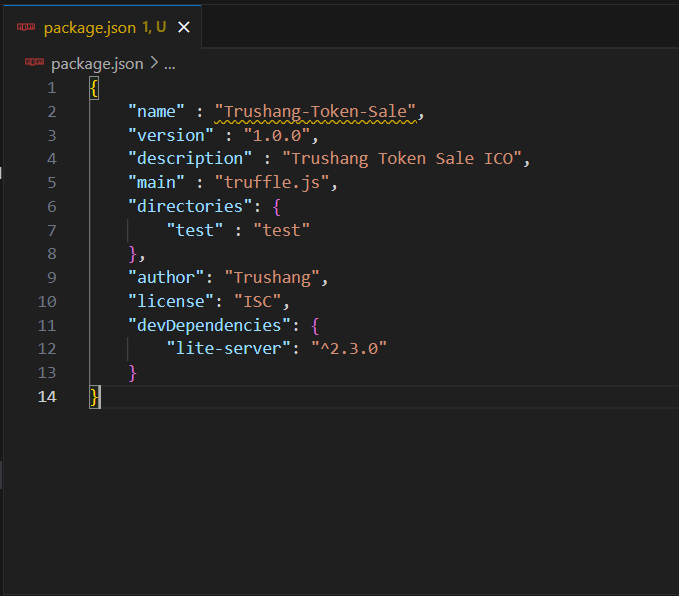


Figure 59: Package. Json file

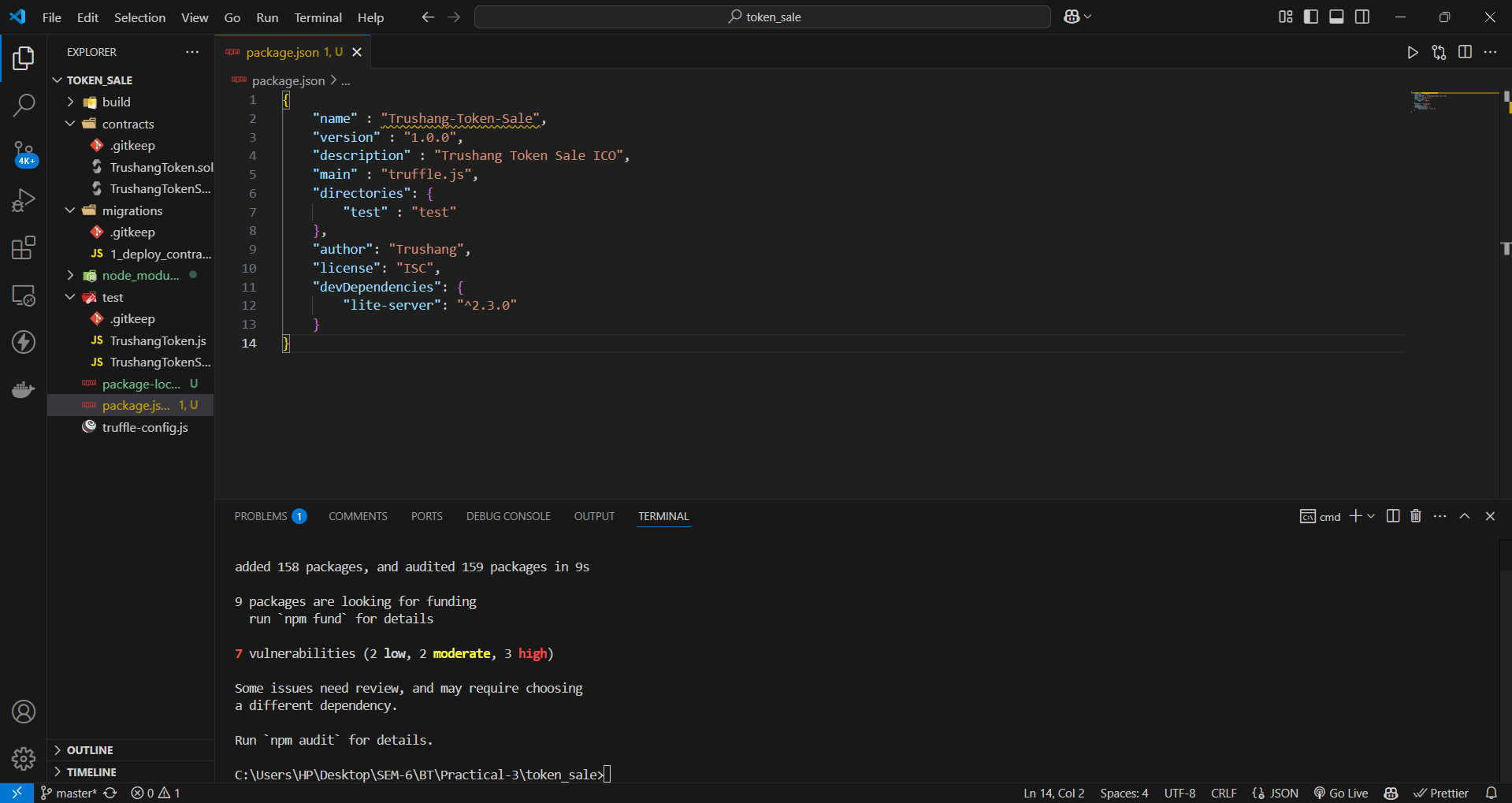


Figure 60:Install dependency

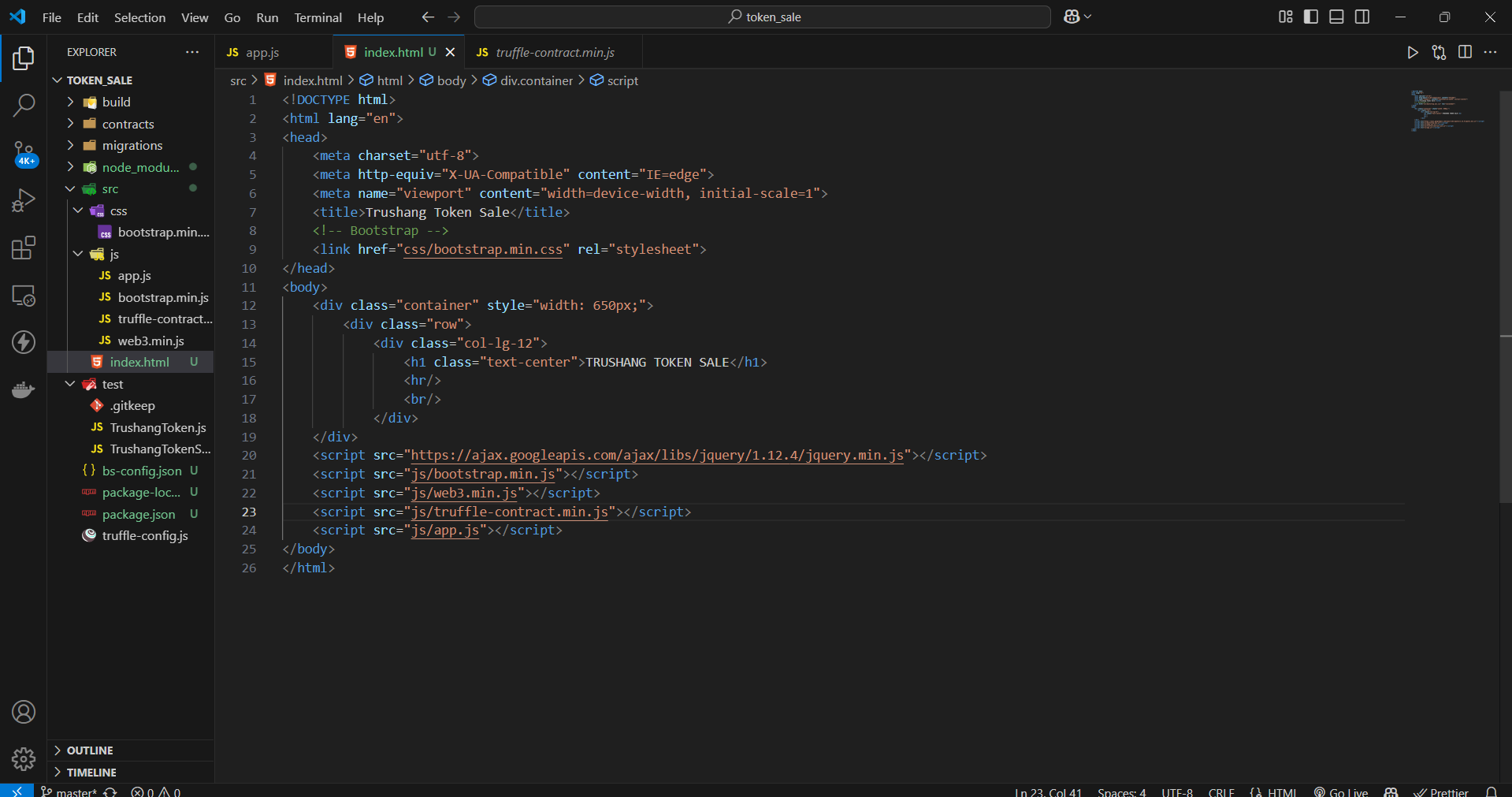


Figure 61:Write UI implementation

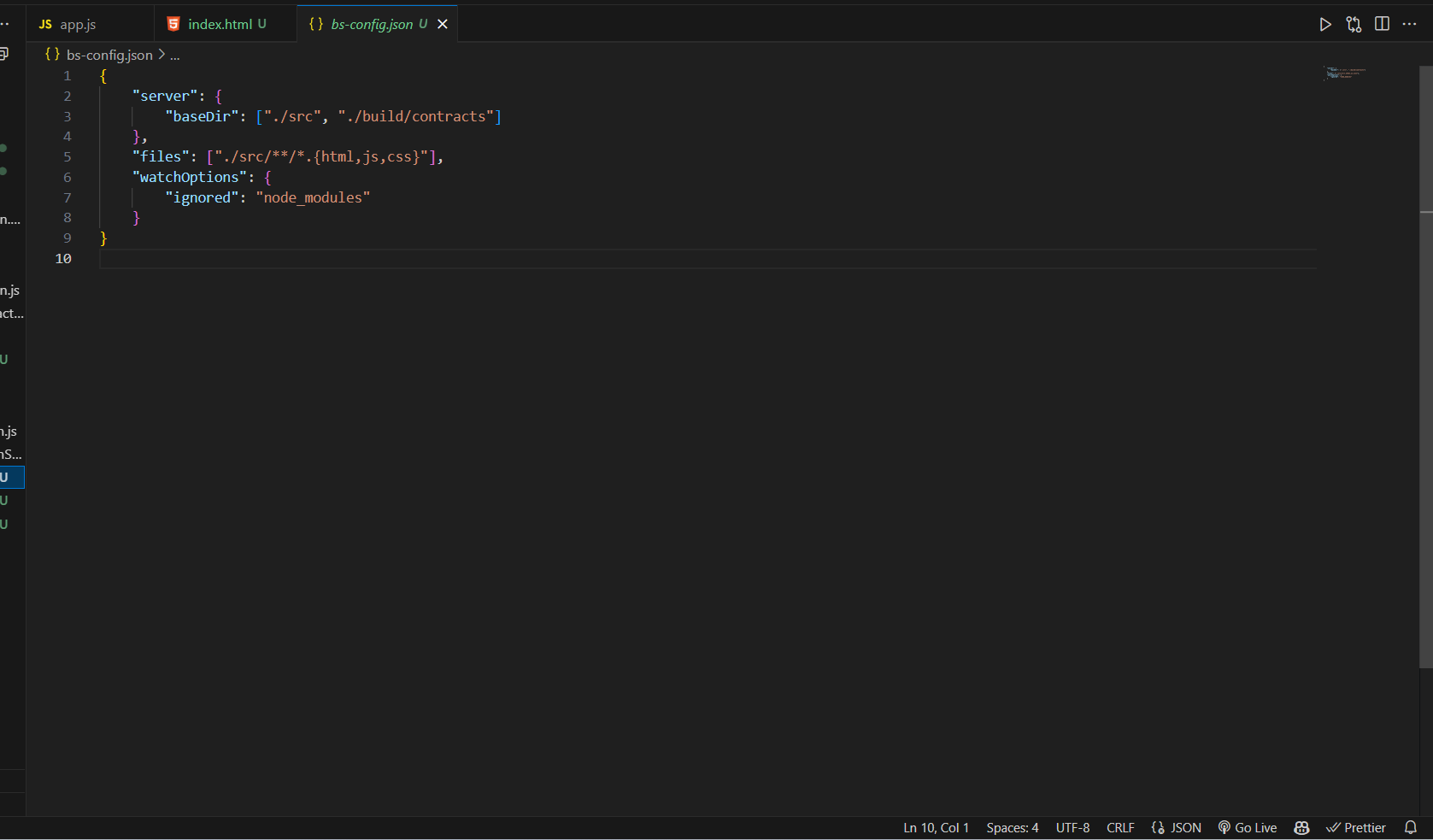


Figure 62:Write bs.config.json

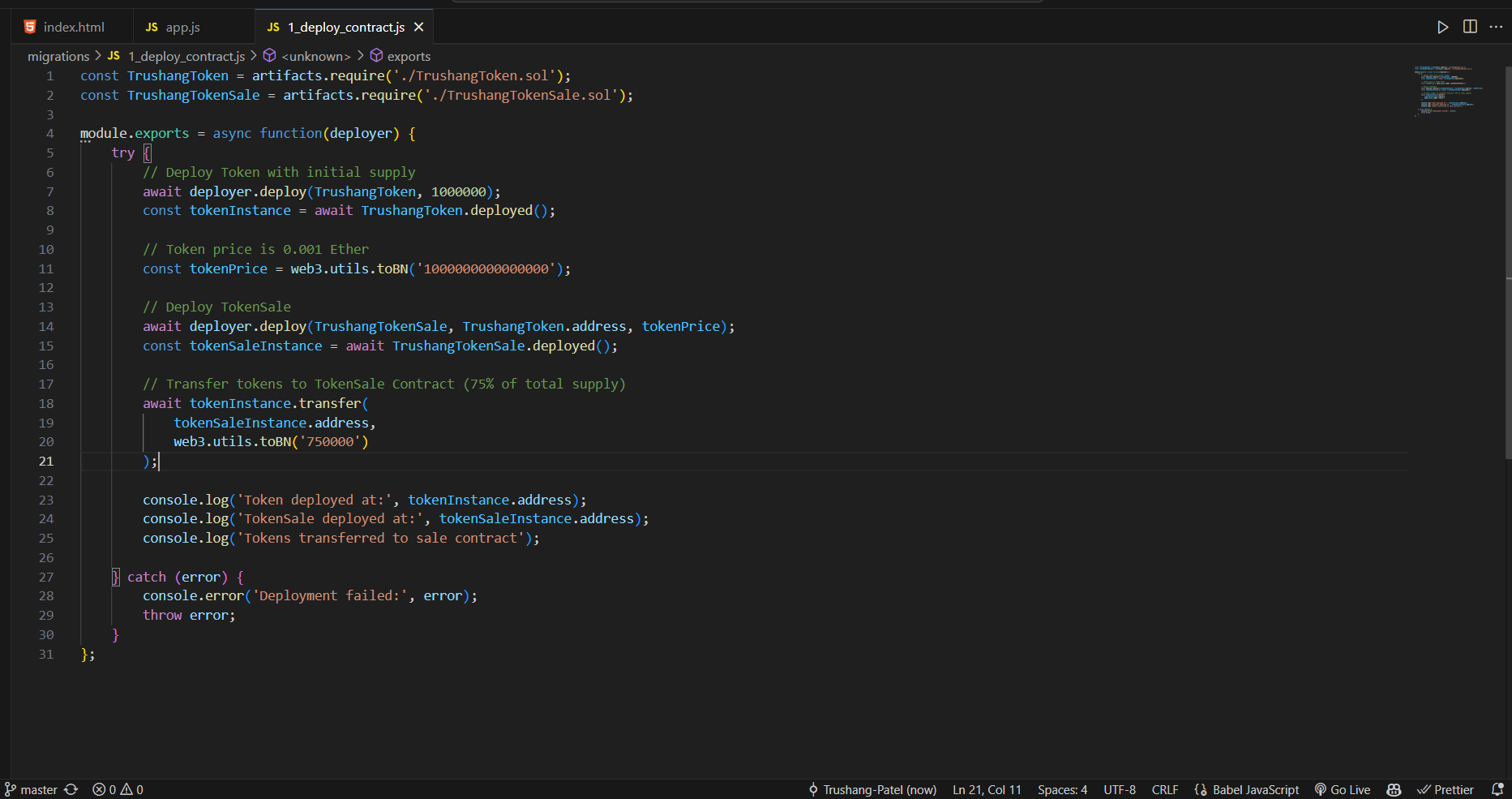


Figure 63:Deploy contract implementation

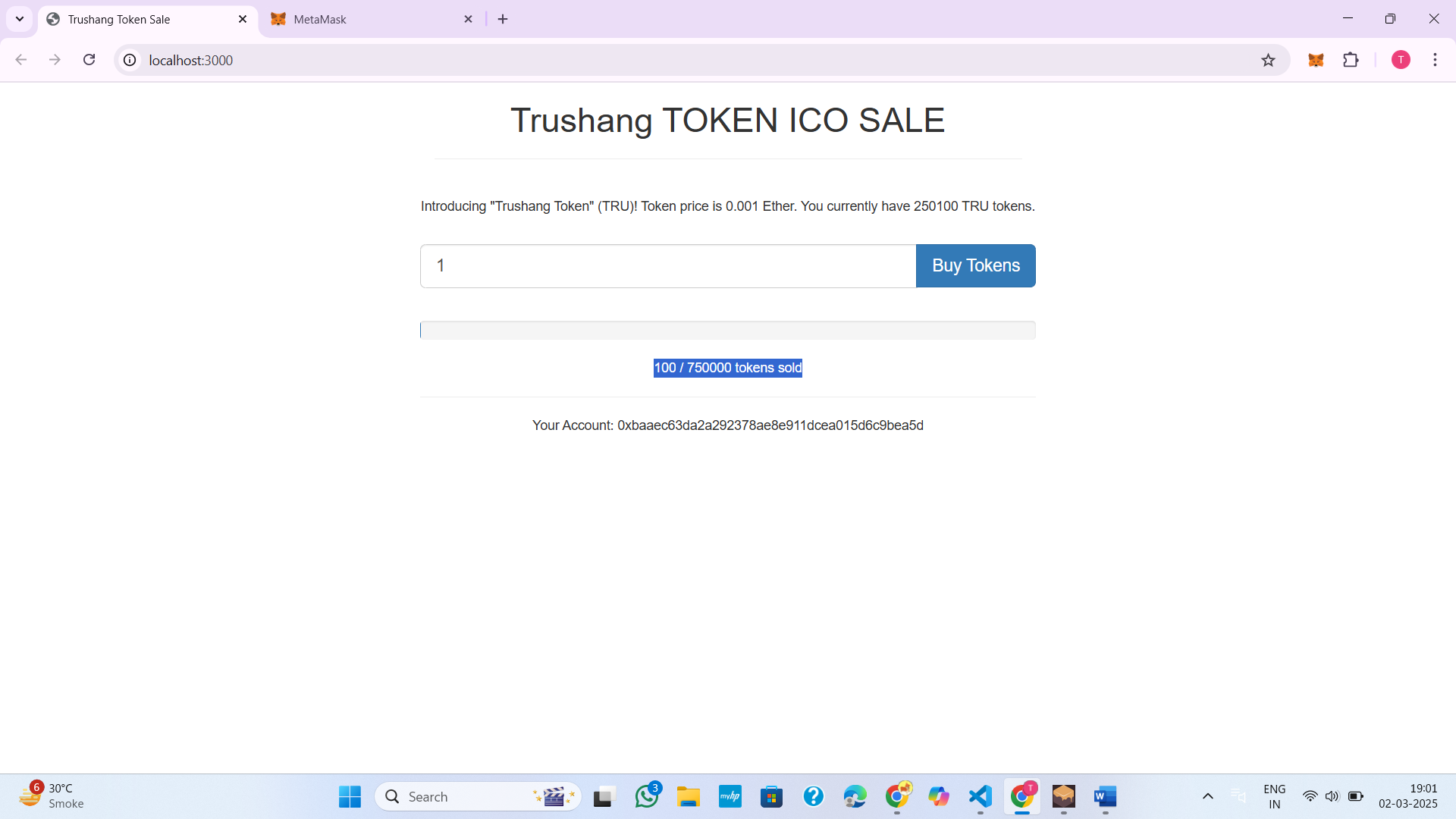


Figure 64:UI of my application



Figure 65:Transaction history

**LATEST APPLICATIONS:**

* Tether USD (USDT)
* USD Coin (USDC)
* Shiba Inu (SHIB)
* Binance USD (BUSD)
* BNB (BNB)
* DAI Stablecoin (DAI)
* HEX (HEX)
* Bitfinex LEO (LEO)
* Maker (MKR)

**LEARNING OUTCOME:**

In this practical, I confidently designed, developed, and deployed my own ERC-20 token on the Ethereum blockchain, integrated it with front-end applications, and understood the broader implications of token economies and blockchain technology. This knowledge provided a solid foundation for further exploration of decentralized finance (DeFi), NFTs, and other blockchain-based innovations.

**REFERENCES:**

1. Udemy: [https://www.udemy.com/course/code-your-own-cryptocurrency/](https://www.udemy.com/course/code-your-own-cryptocurrency/?srsltid=AfmBOopjvZsDJ6uJbYHew5nbOgyWiNRw0w2-x5fe6z_D53kyYdmjOQA3)
2. GitHub: <https://github.com/Trushang-Patel/token_sale>