BlockchainFoodOrder Smart Contract Deployment and Interactions Guide

Shiden Network is a multi-chain decentralized application layer on Kusama Network. Shibuya is the Shiden's parachain testnet with EVM functionalities. We choose it for deployment as Shiden supports EVM, Wasm, and Layer2 solutions.

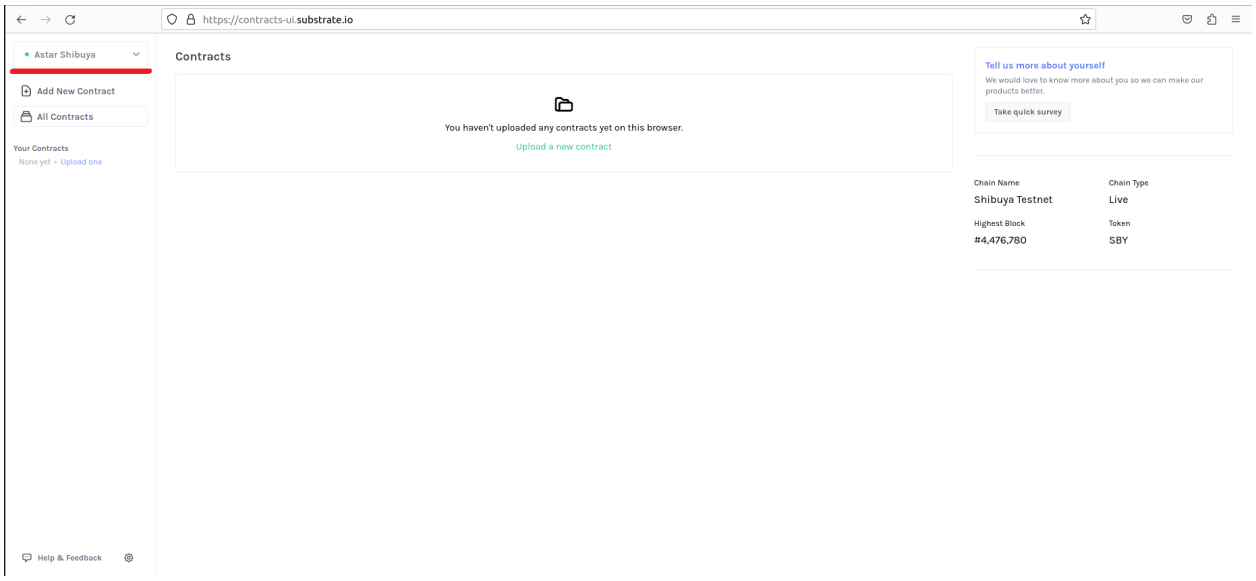
Contracts-UI is a web application for deploying Wasm smart contracts on Substrate chains. Polkadot.js is an effort to provide a collection of tools, utilities and libraries for interacting with the Polkadot network from JavaScript.

We prepared this shared document with screenshots to illustrate the flow of deploying the contract then step-by-step flow of ordering food on the blockchain. It is generally recommended to use the more user friendly Contracts UI over the PolkadotJS app for ink! deployments and interactions.

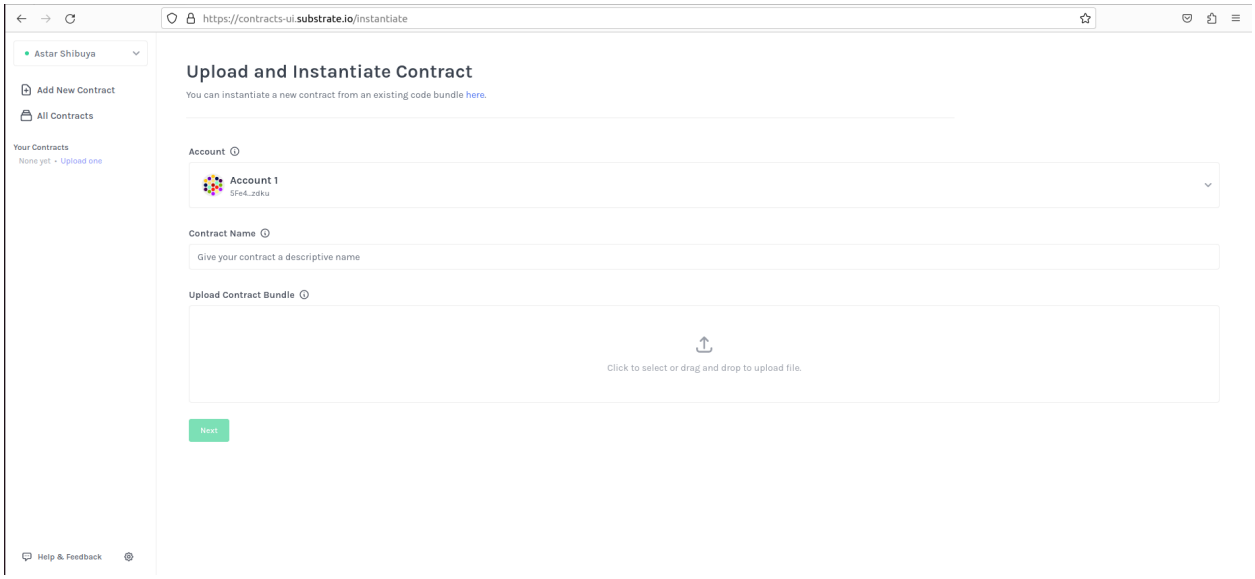
Working with Contracts-UI

Upload and instantiate

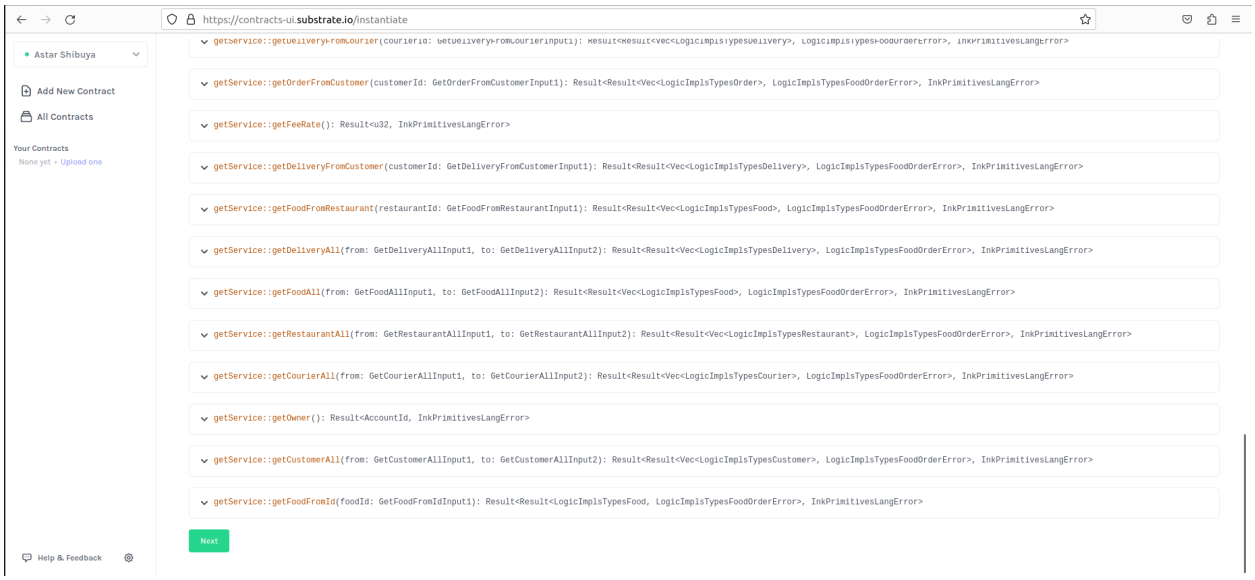
1. Change the network to Astar Shibuya.



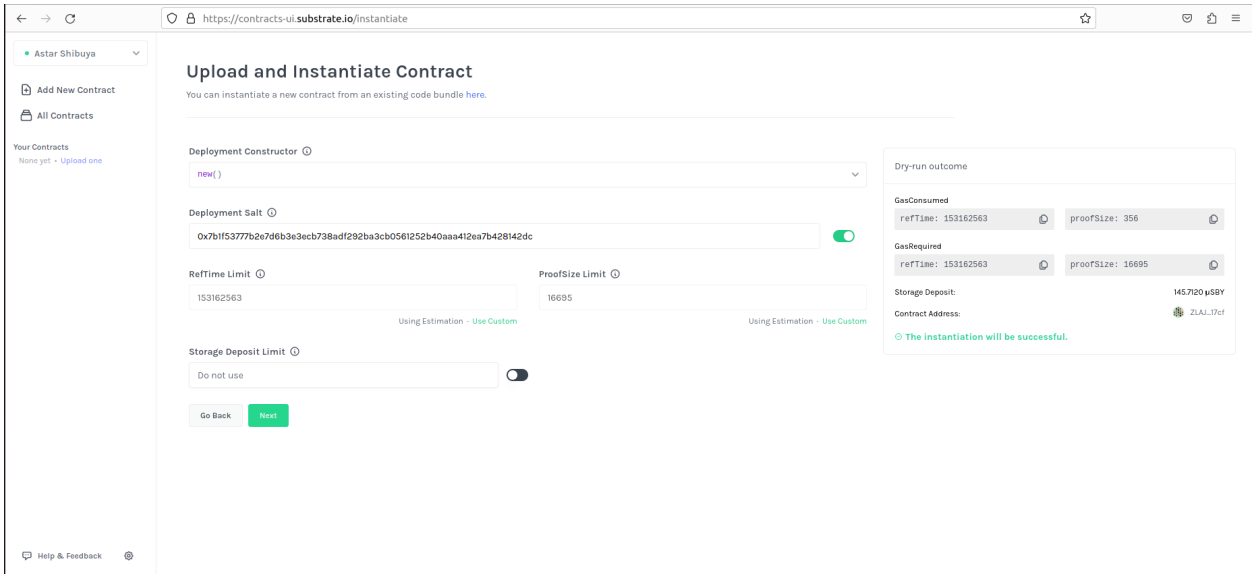
2. Click the "Upload a new contract" button.



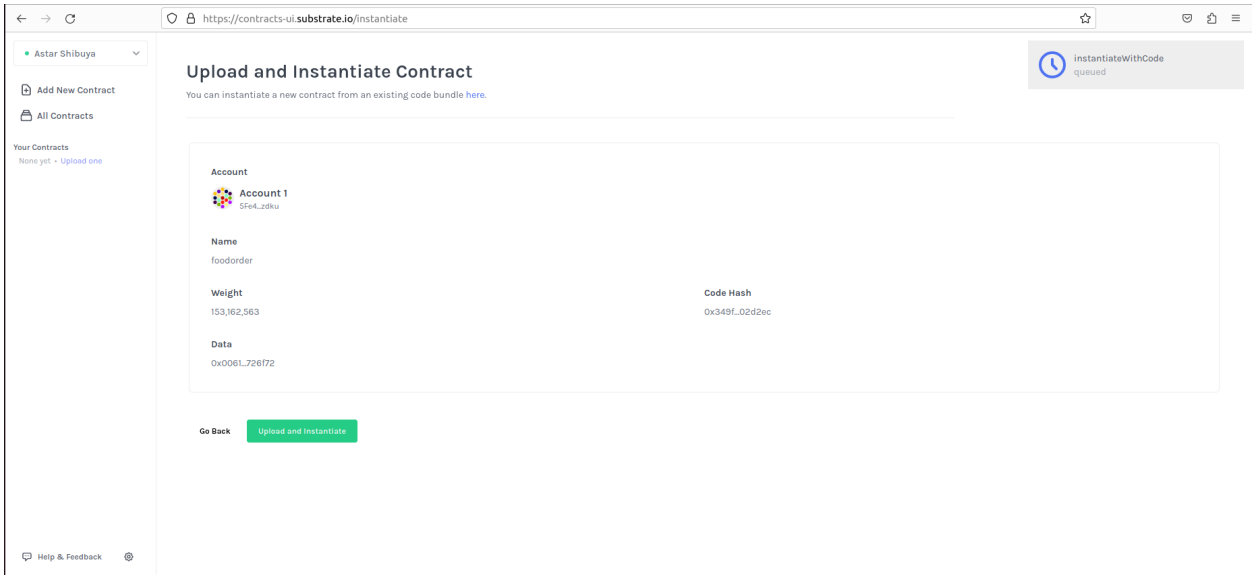
3. Drag and drop the “foodorder.contract” file and click the “Next” button.



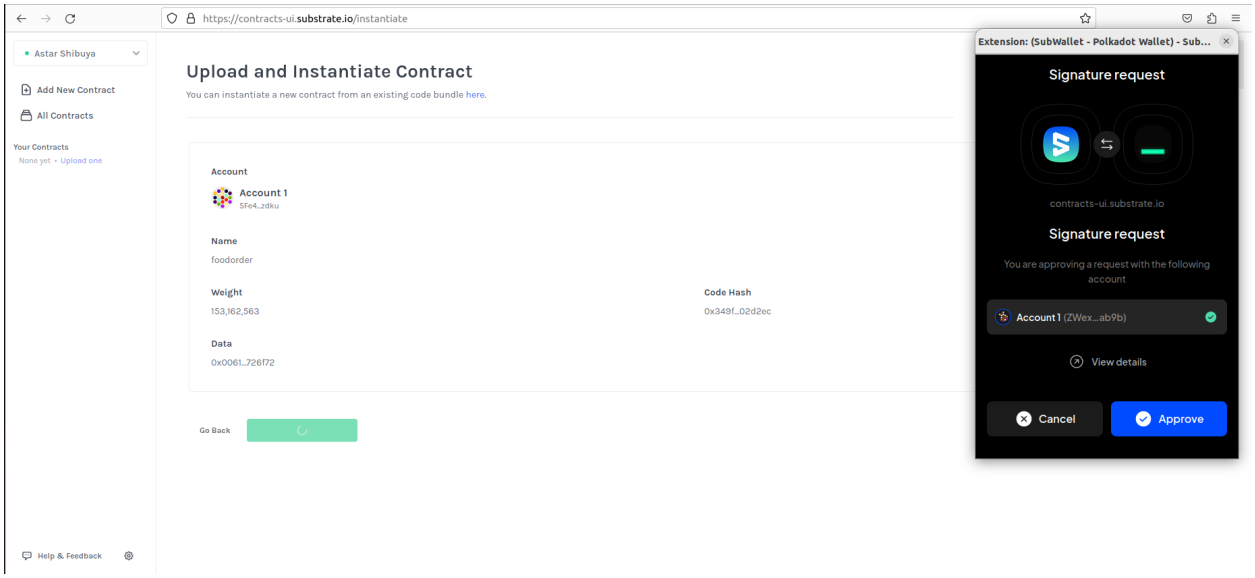
4. Click the “Next” button after checking the messages.



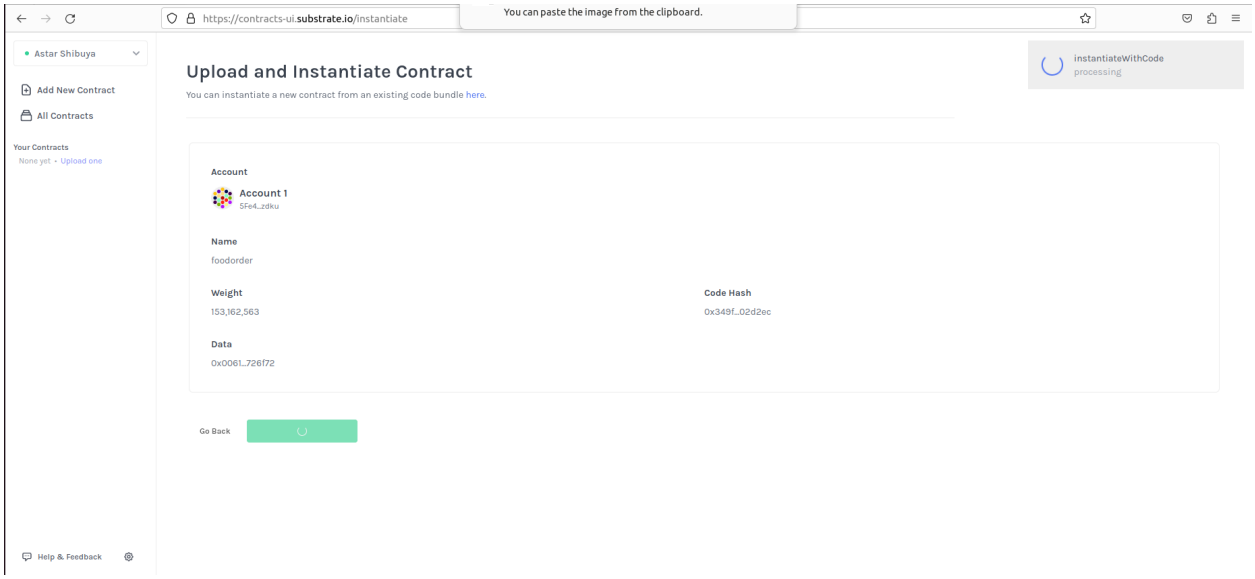
5. Click the “Next” button after checking the constructor message instantiates the contract.



6. Click the “Upload and Instantiate” button to finalize the uploading and instantiating the contract.



7. You can see the pop-up window and click the “Approve” button.

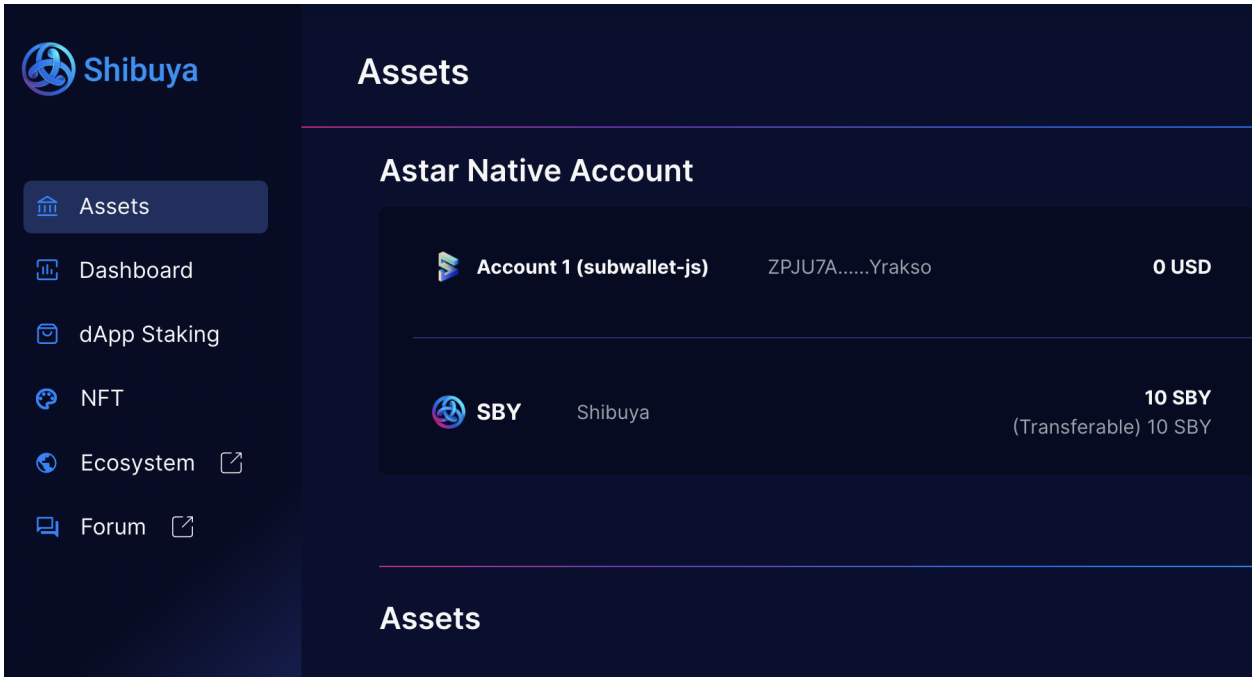


8. Now, the BlockchainFoodORder smart contract has successfully been deployed on Shibuya via Contracts-UI at address: YezJmtfEtFowEBto53VCpkLGhvFXtpe88frPW1q1z7Y2jUd, you can interact with the contract’s messages.

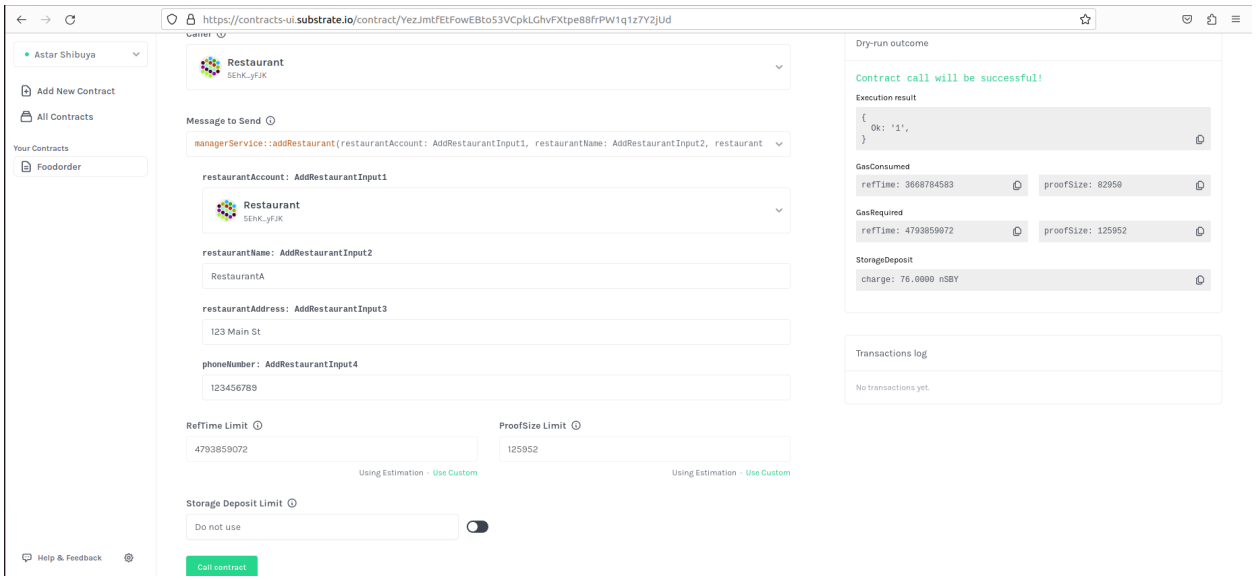
Interact with the smart contract

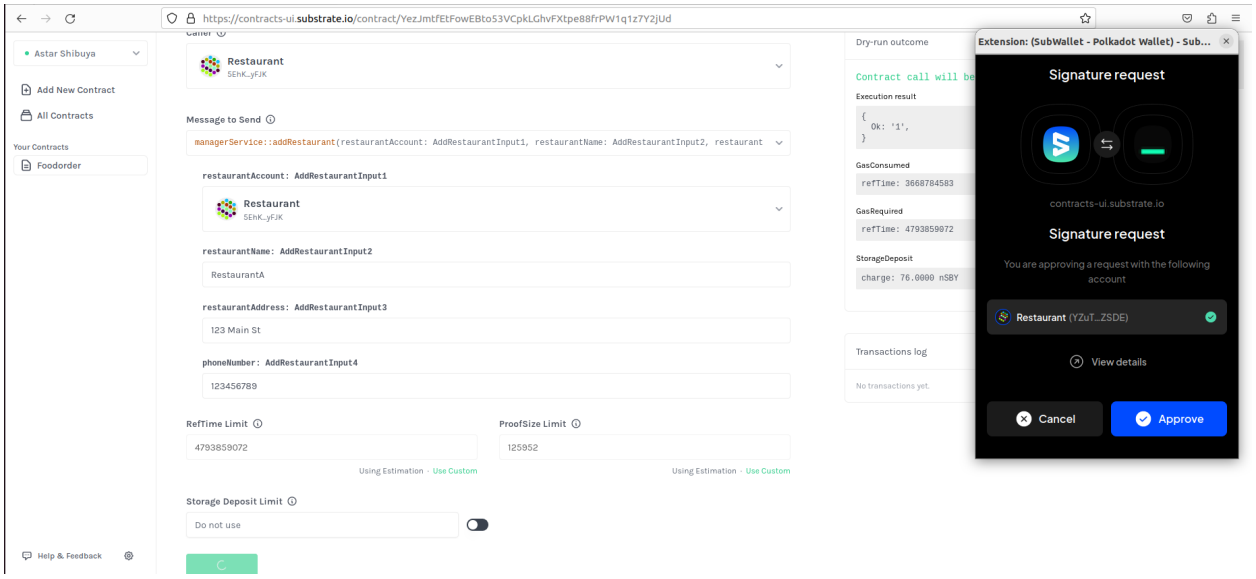
1. Get free SBY tokens from a faucet.

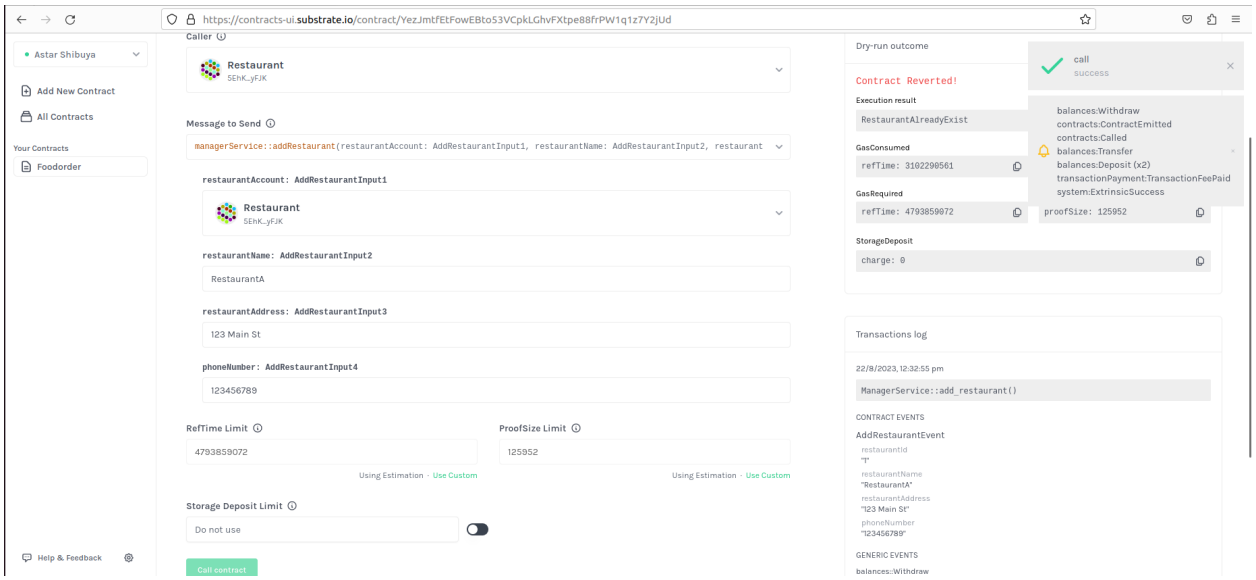
Before you are trying to execute a transaction, you will need to get native tokens from a faucet.



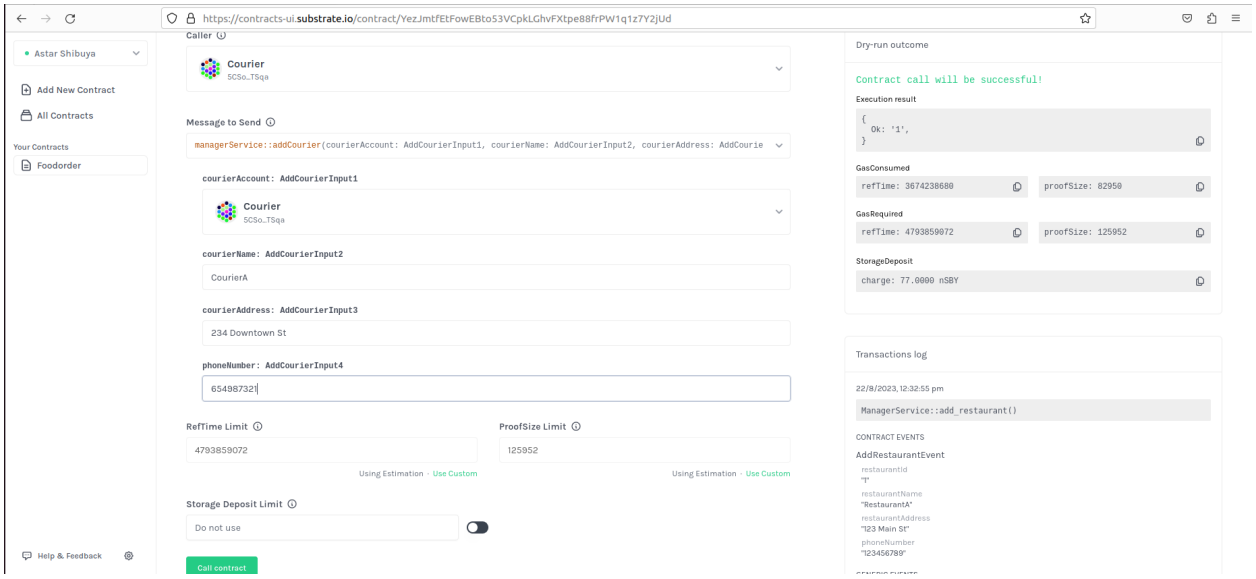
2. Restaurant registers itself into the smart contract storage.

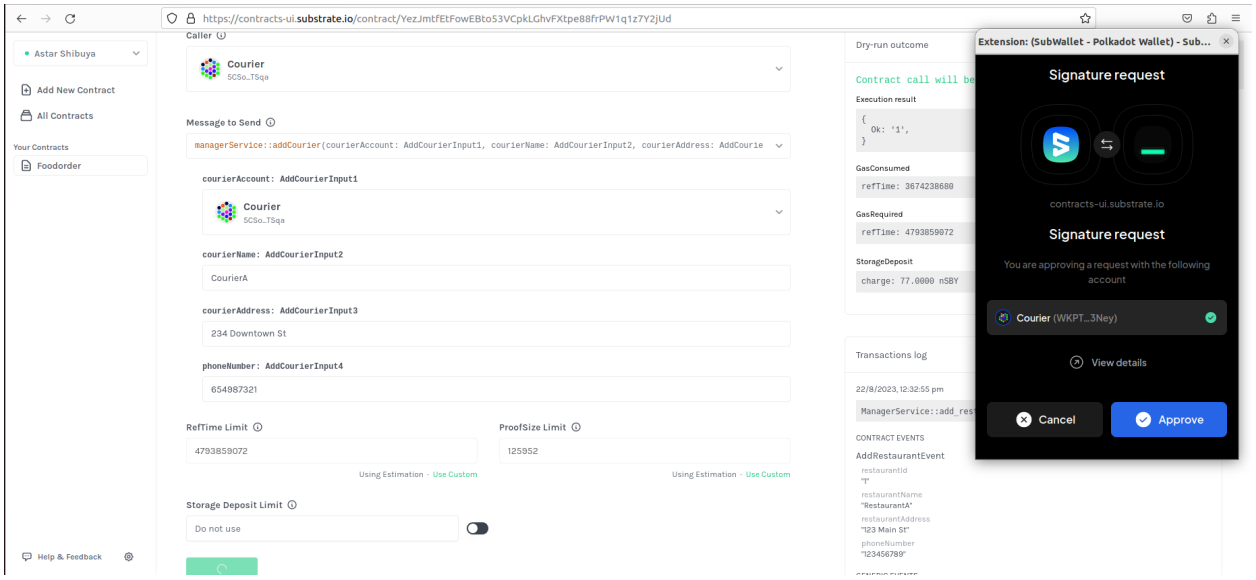




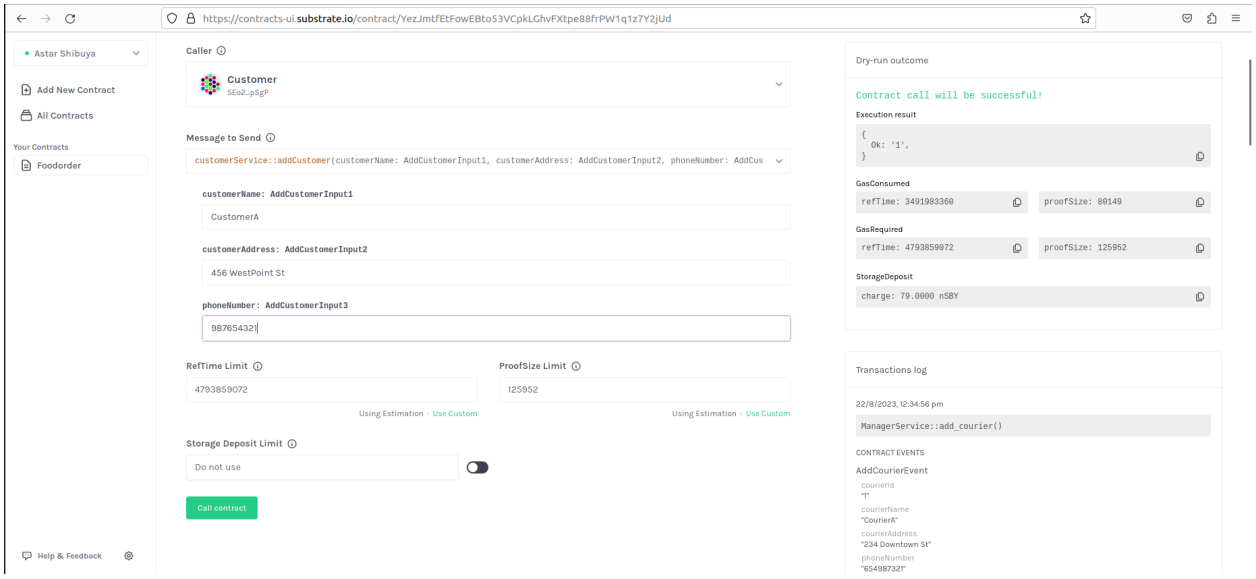


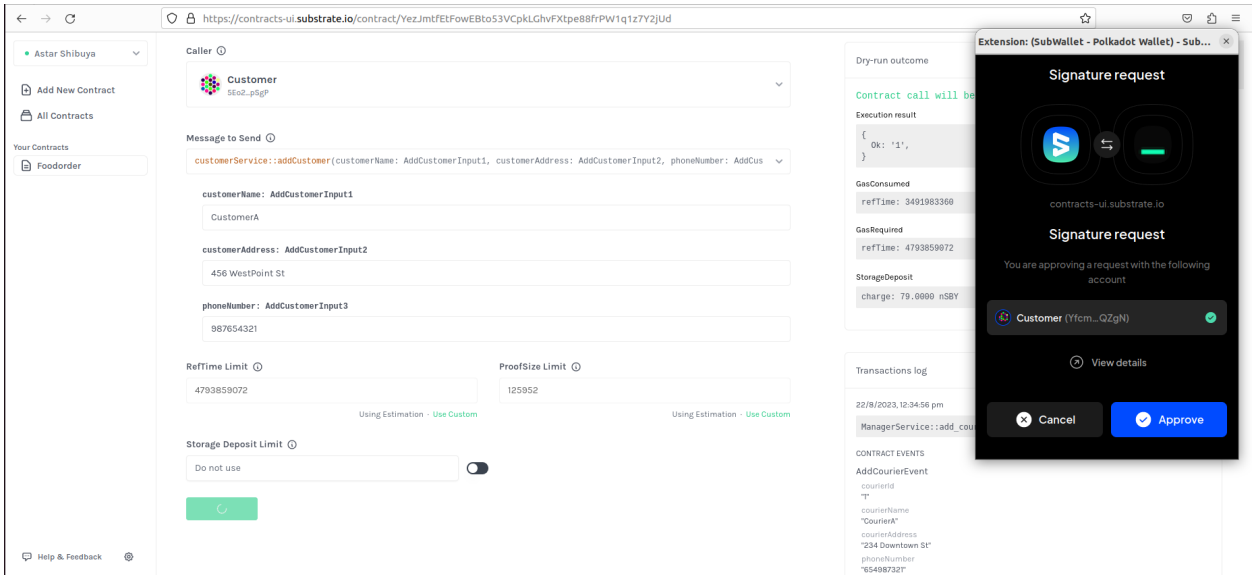
3. The Courier registers itself into the smart contract.



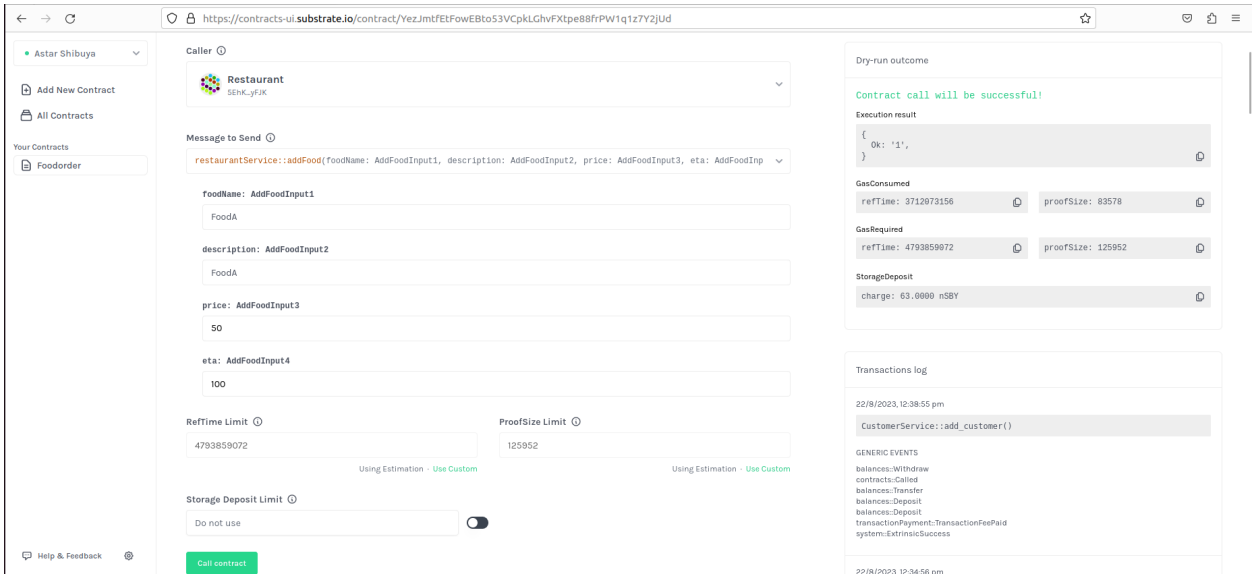


4. A Customer registers itself into the smart contract.

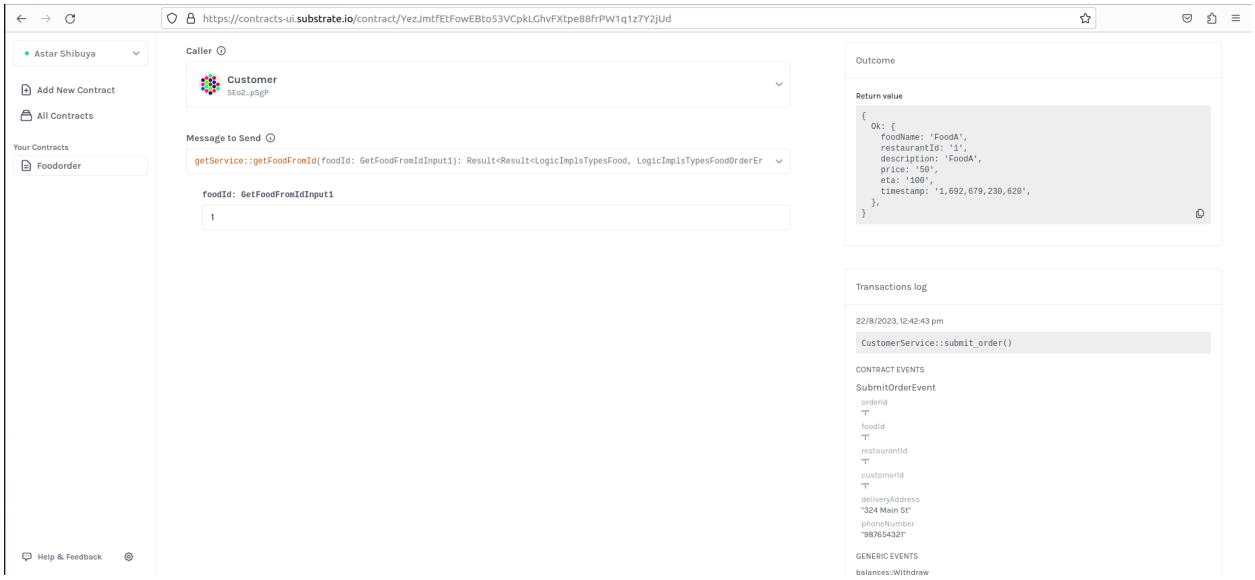




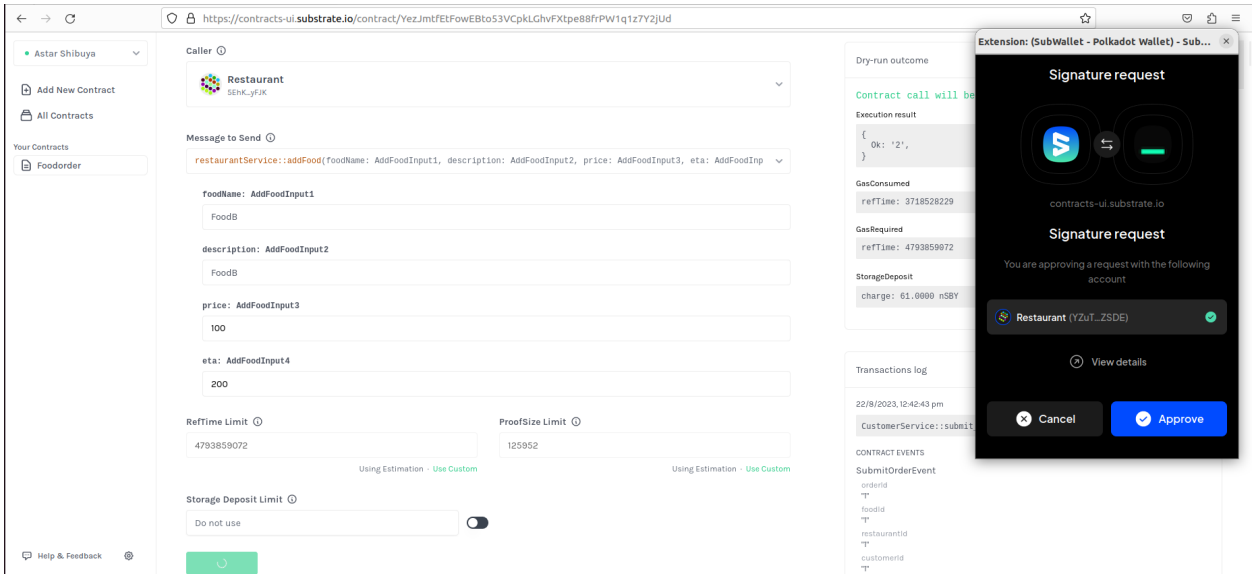
5. The Restaurant adds one food.



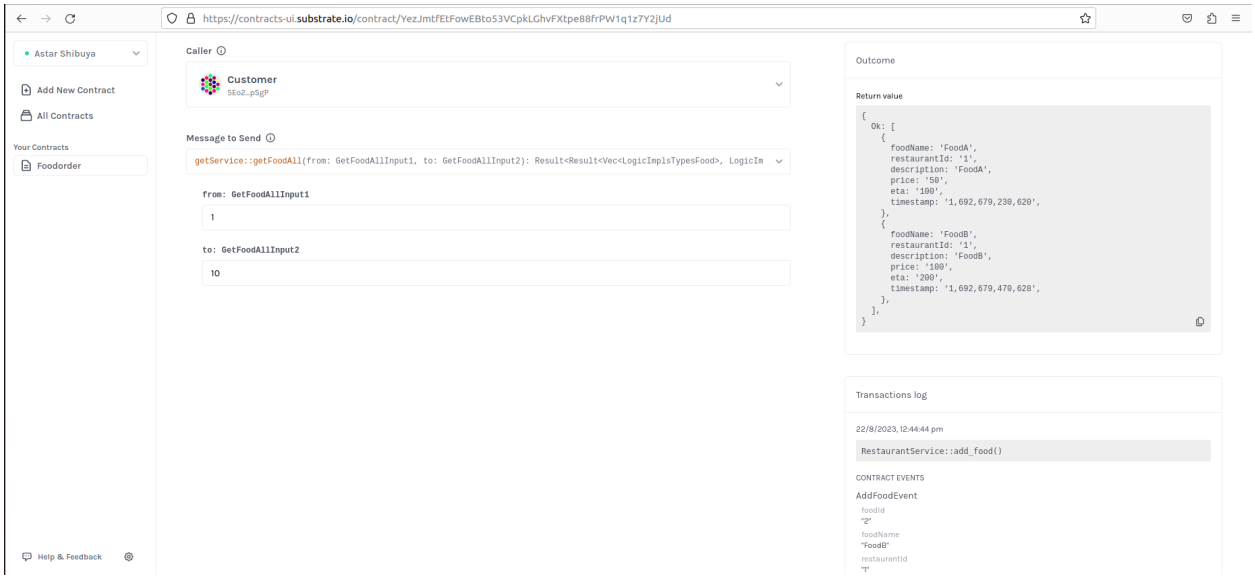
6. Check the “getFoodFromId” message’s result.



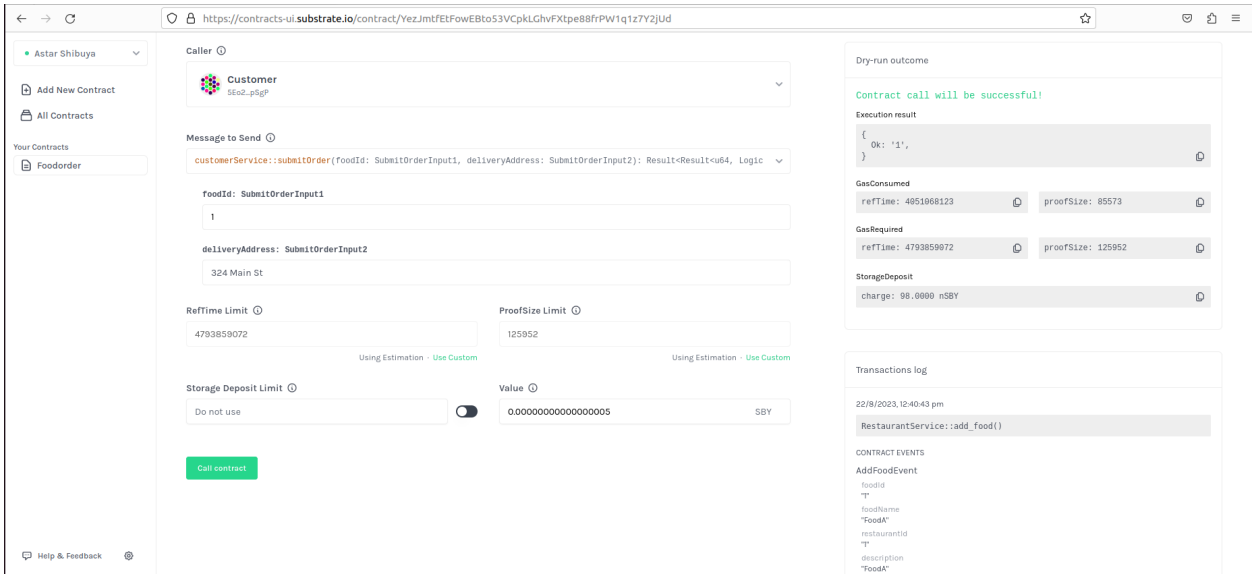
7. The restaurant adds one more food.



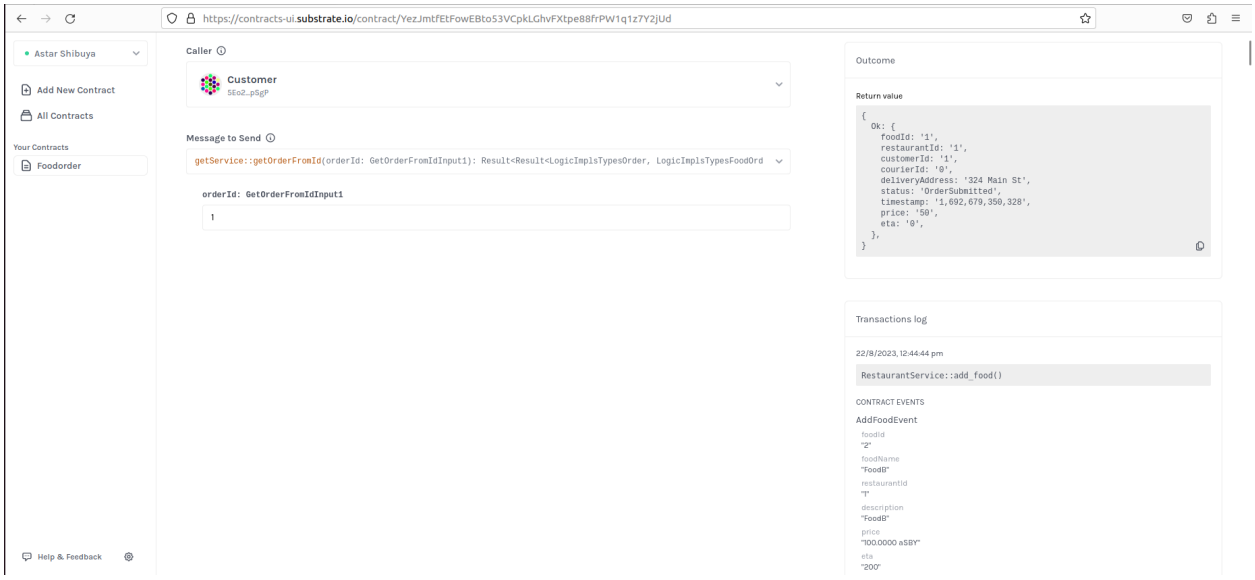
8. Check the “getFoodAlll” message’s result.

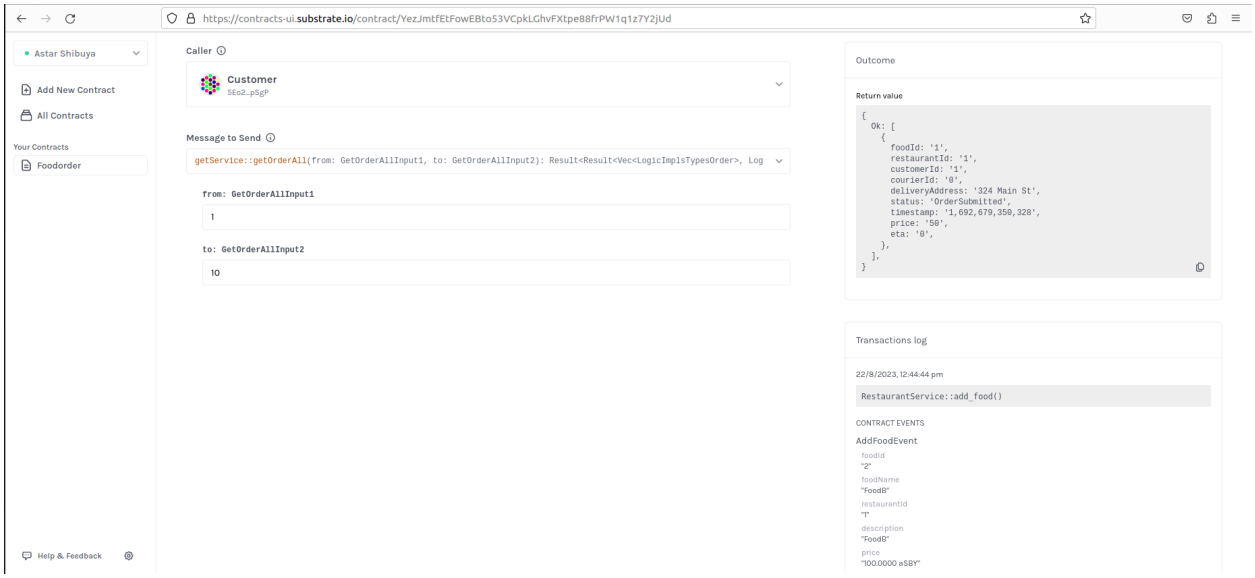


9. The customer submits an order.

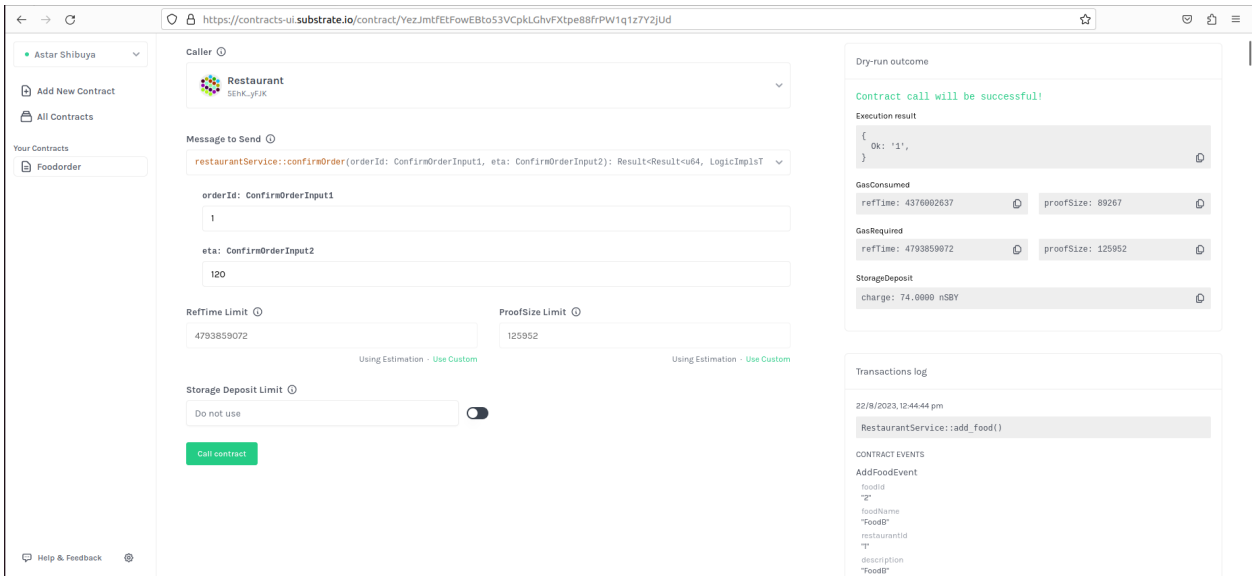


10. Check the order status.

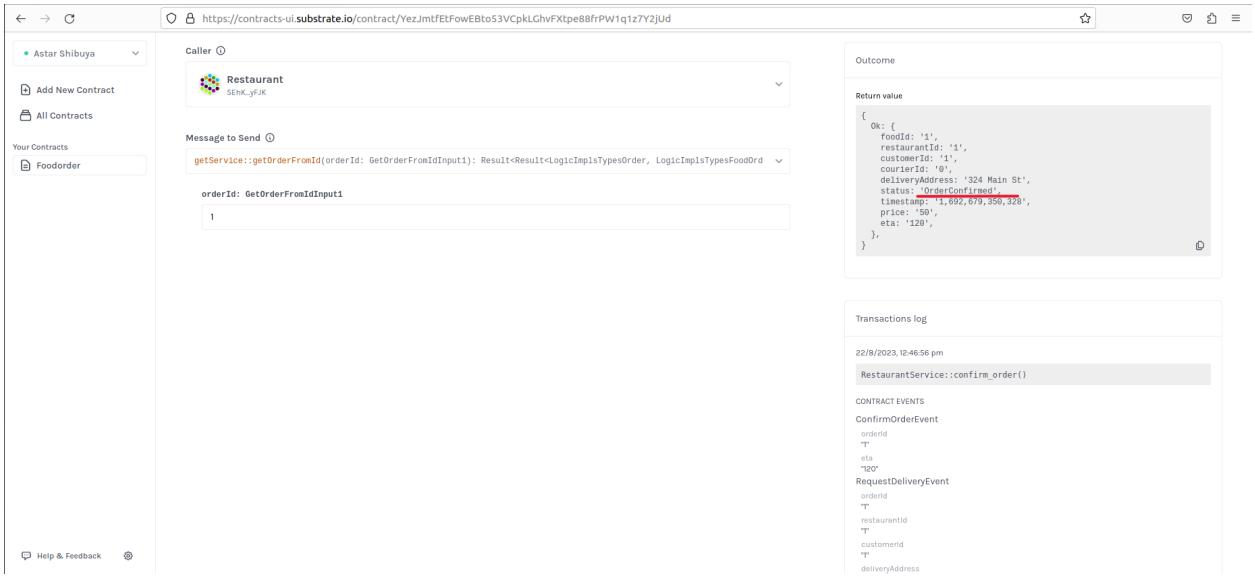


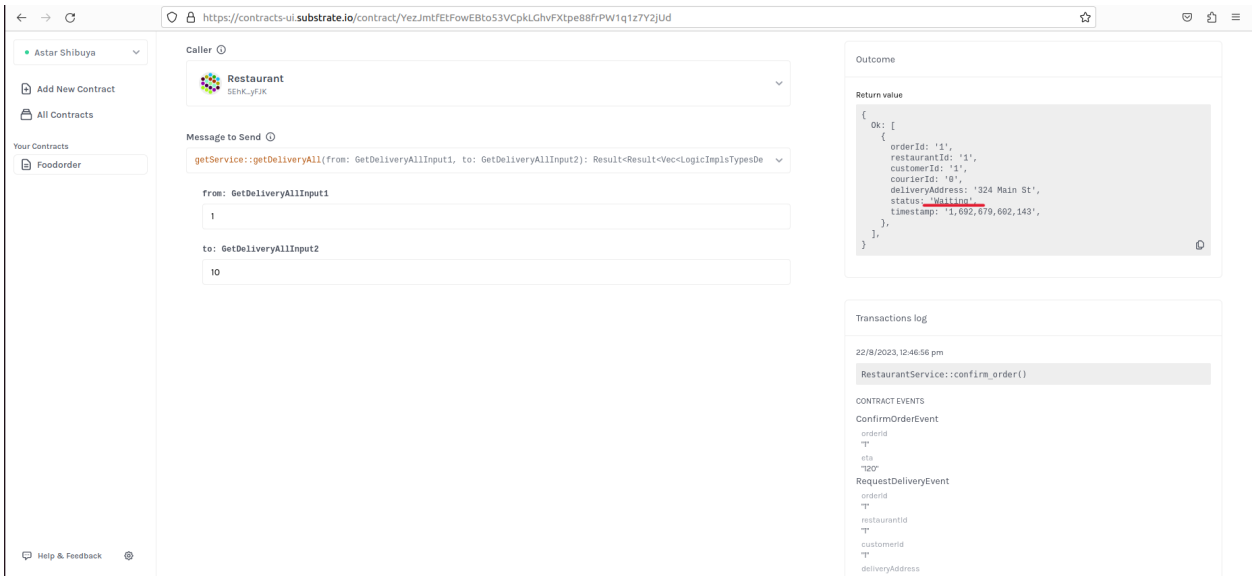


11. The restaurant confirms an order before starting the cook and calls a courier to deliver the order after eta deadline.

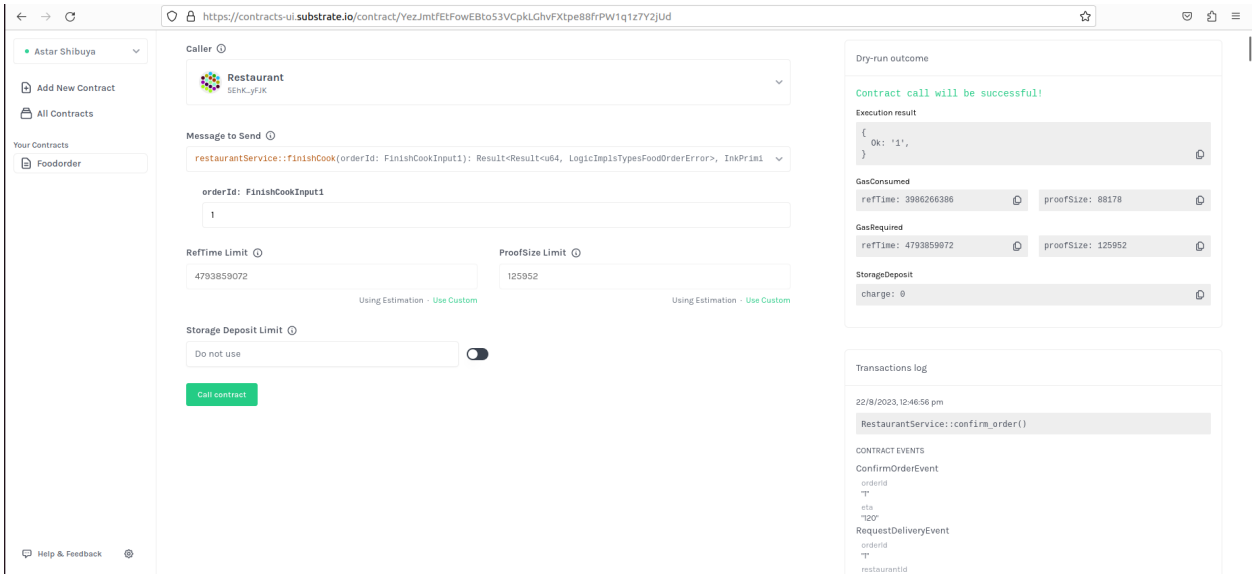


12. Check the order status and delivery status.

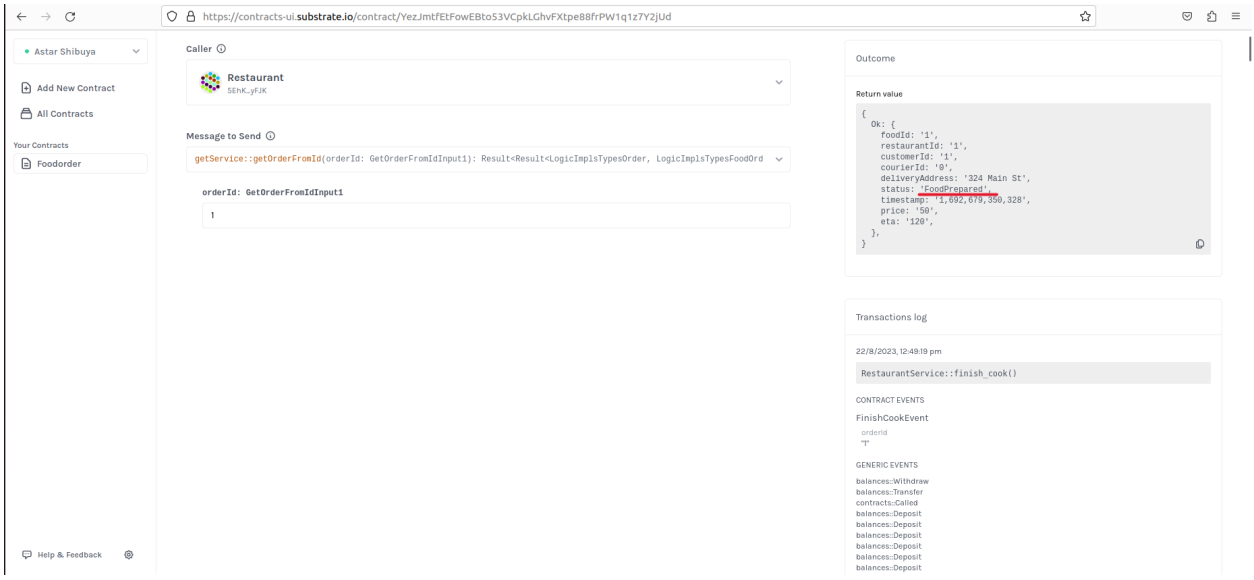




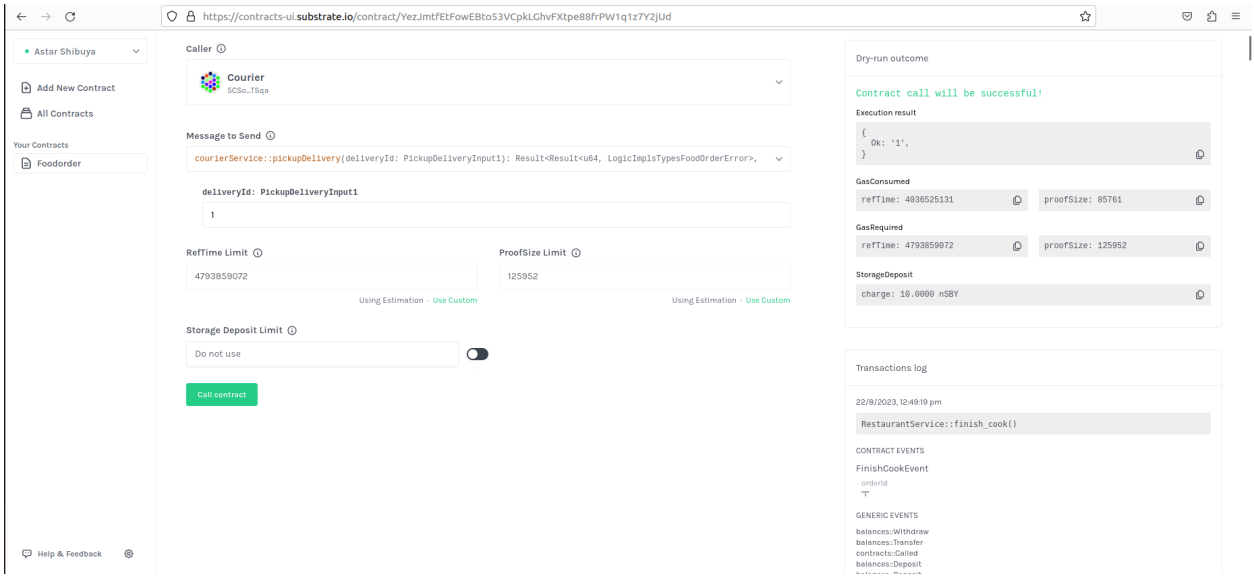
13. The restaurant finishes the cooking of the submitted order.



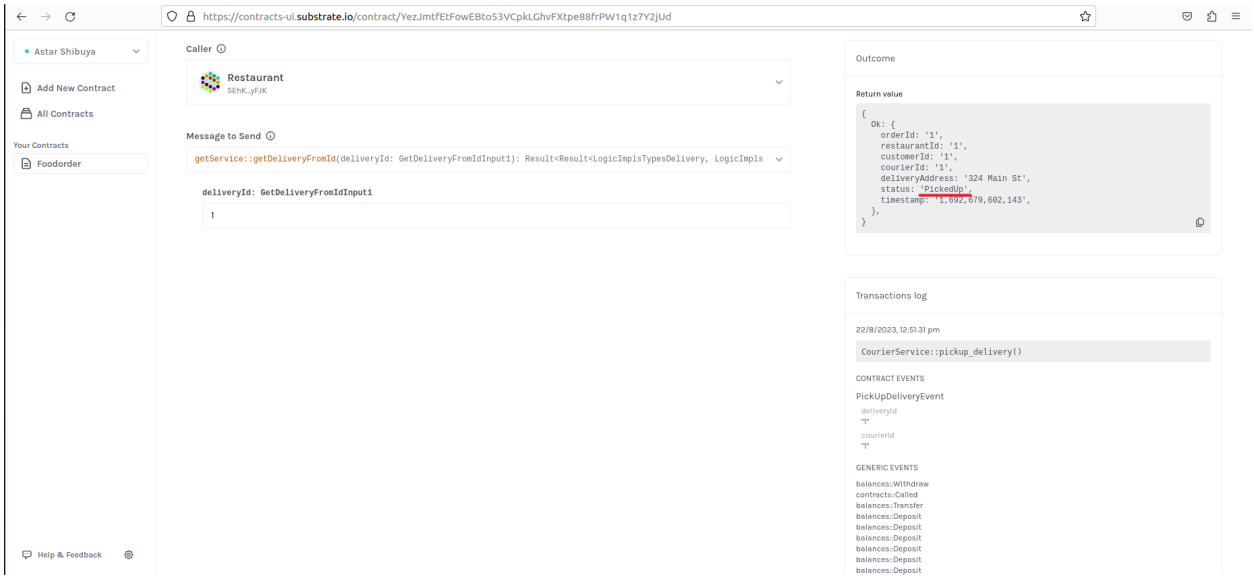
14. Check the order status.



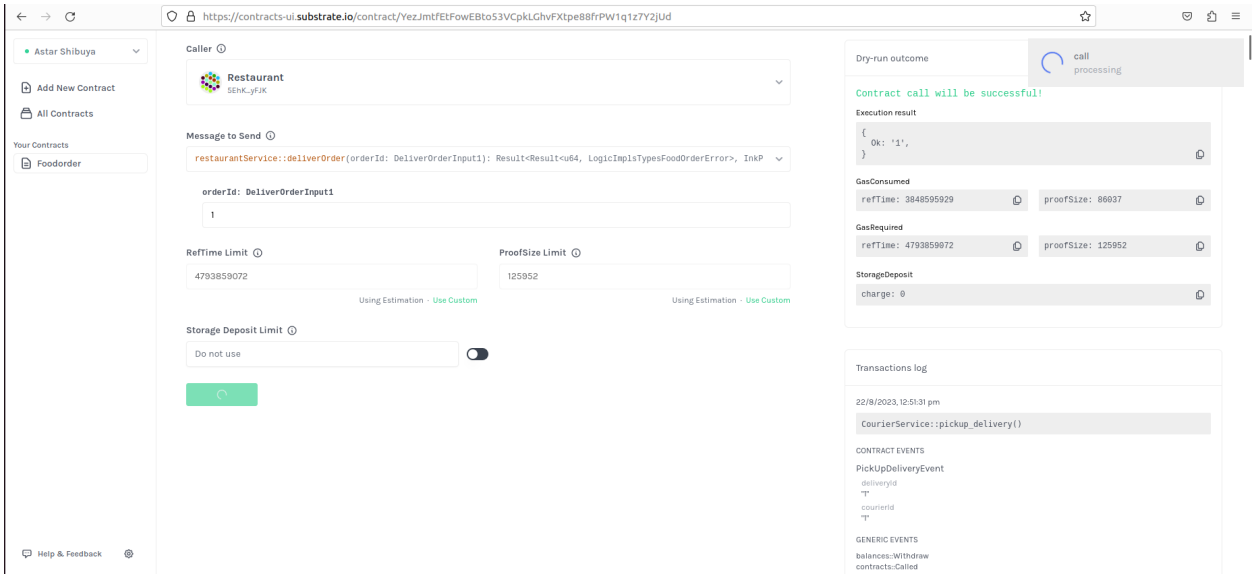
15. The courier picks up the delivery after food is prepared.



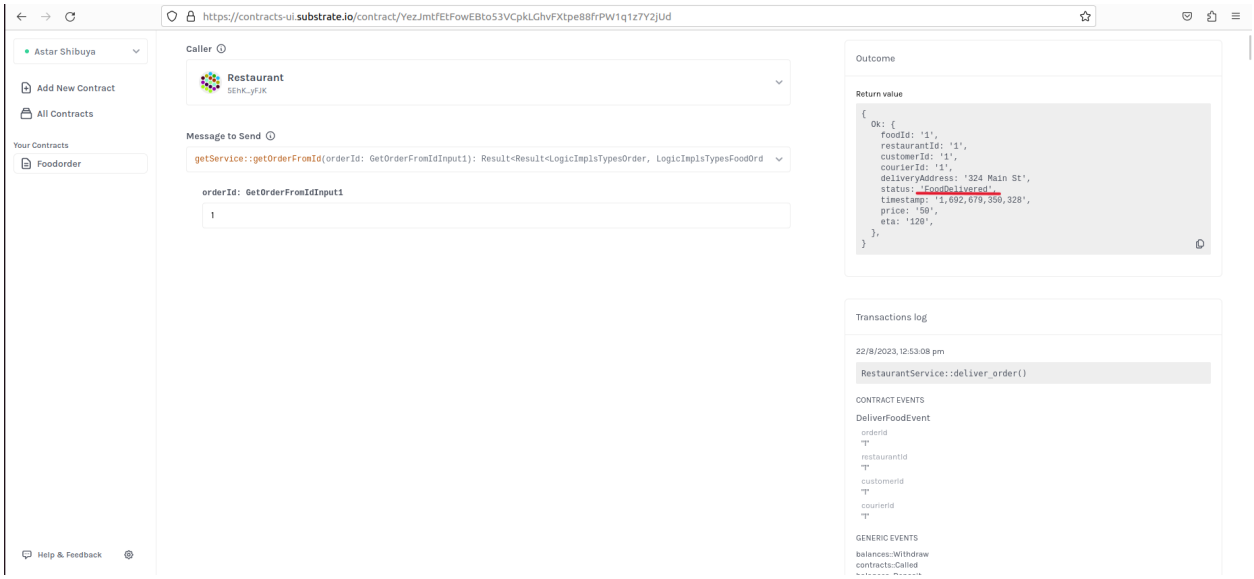
16. Check the delivery status.



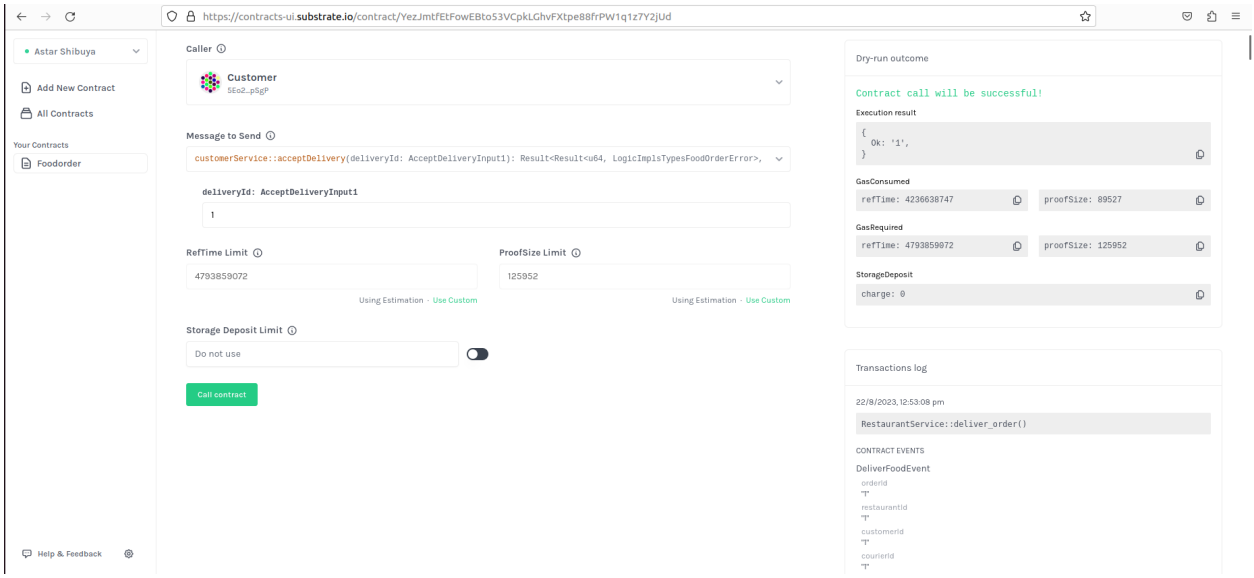
17. The restaurant change the order status to “FoodDelivered”.



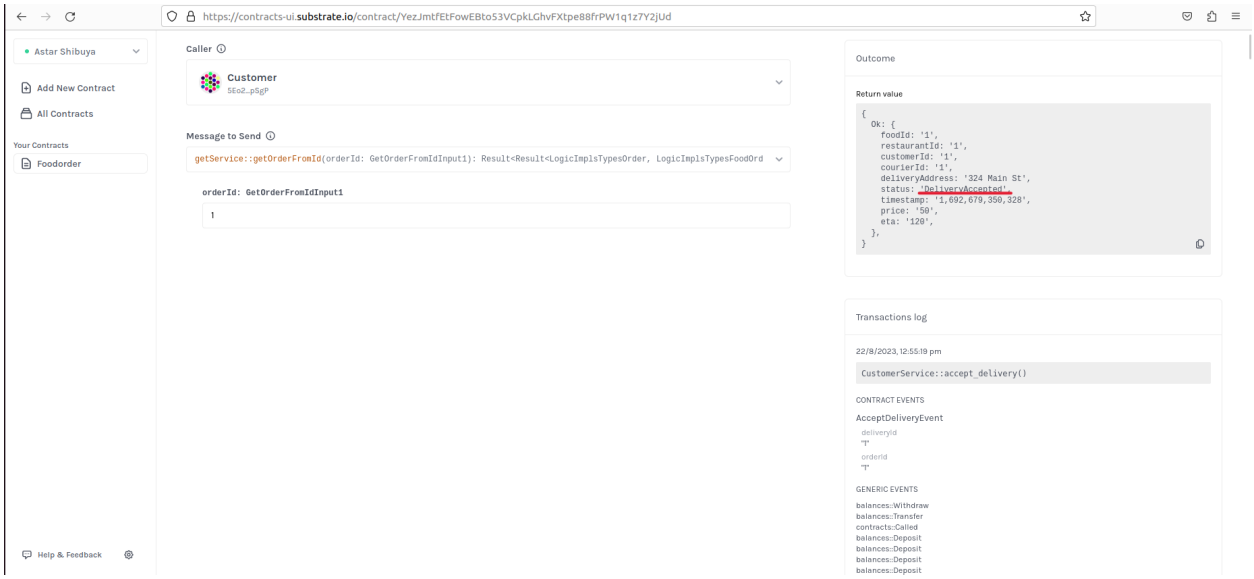
18. Check the order status.



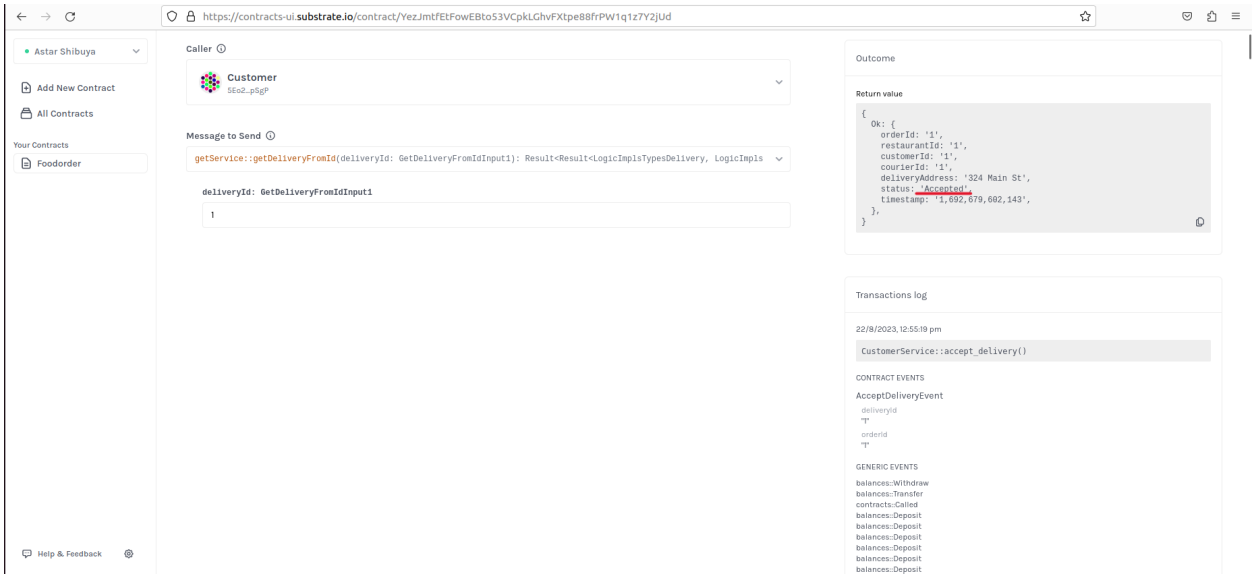
19. The customer accepts the delivery.



20. Check the order status.

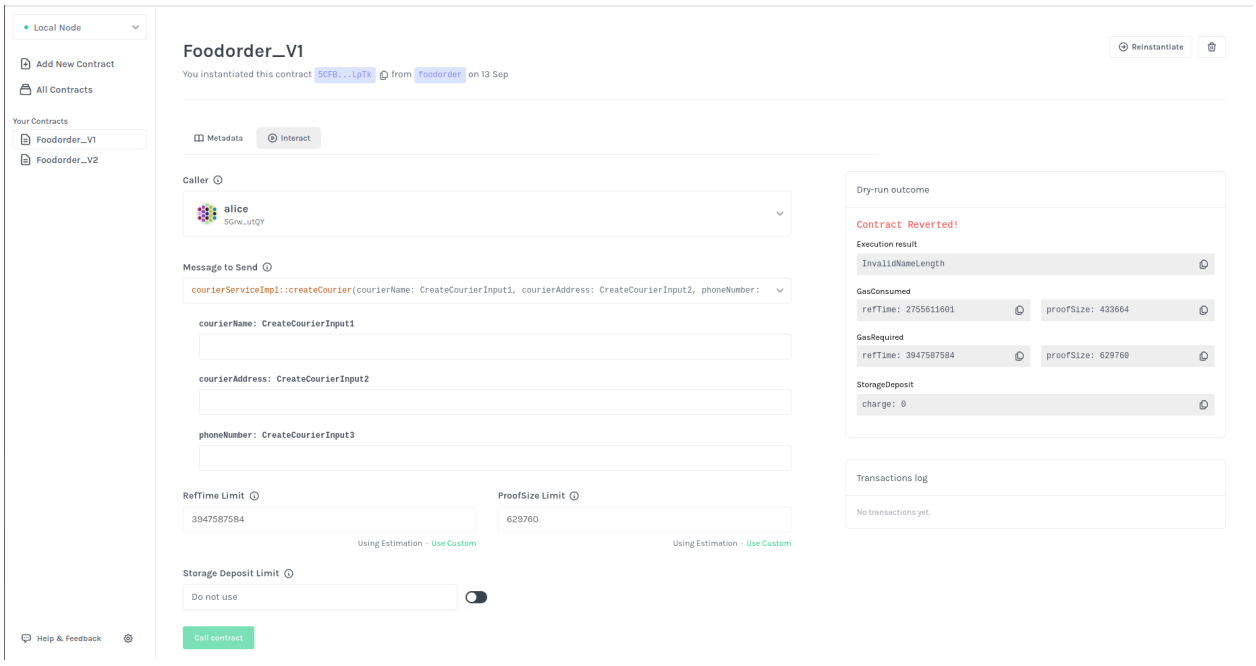


21. Check the delivery status.

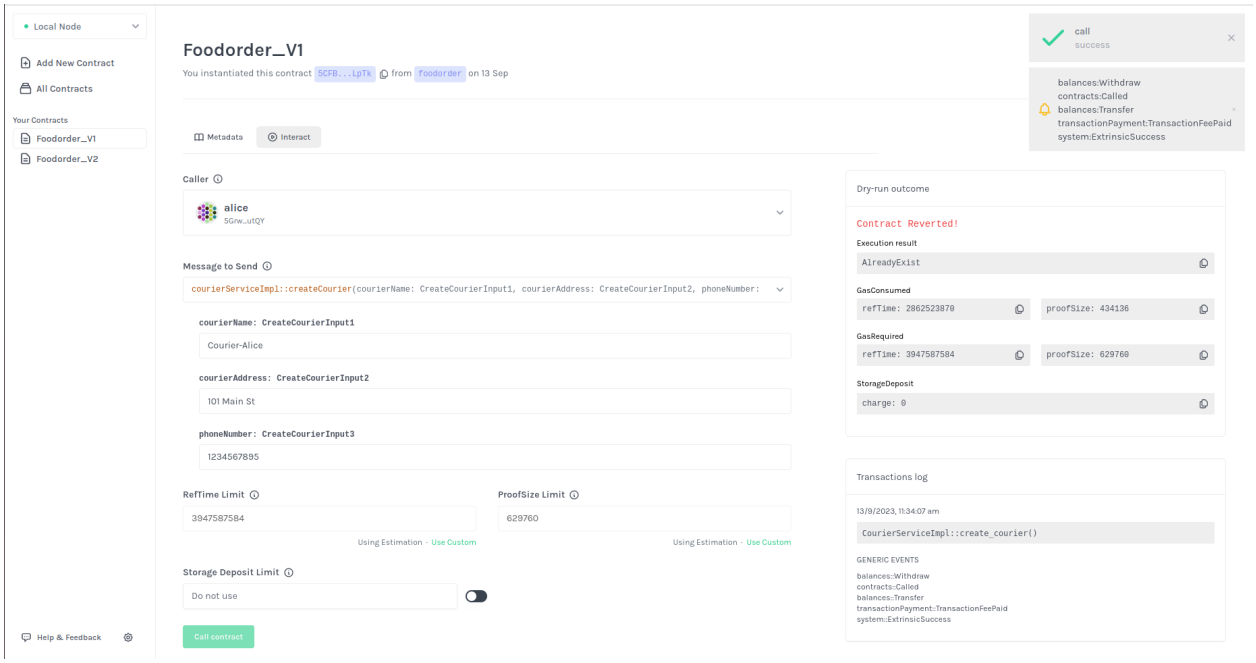


Capturing smart contract upgrades

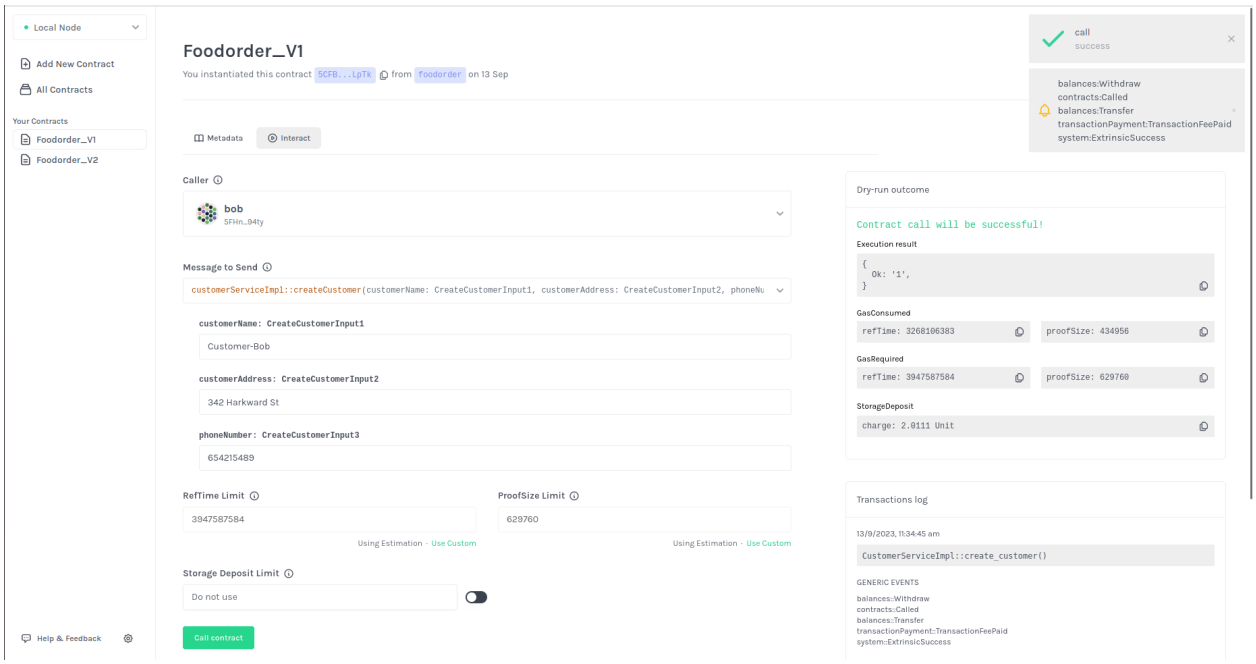
1. Instantiate FoodOrder_V1 contract.



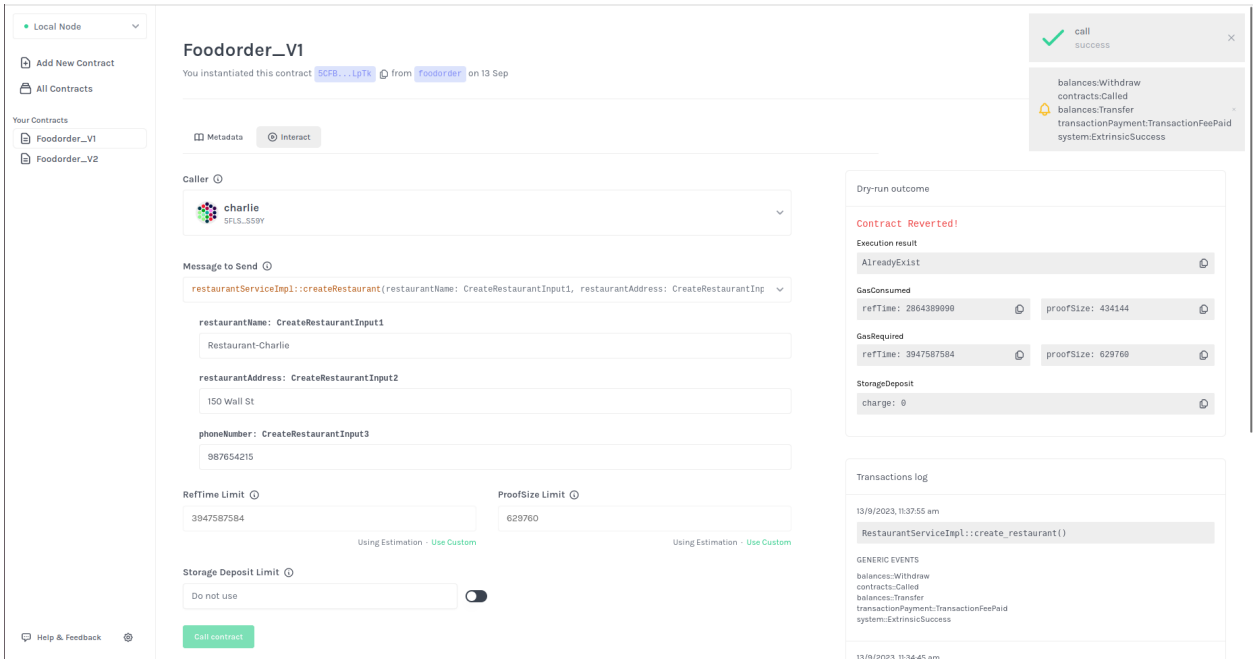
2. Create a courier in FoodOrder_V1 contract.



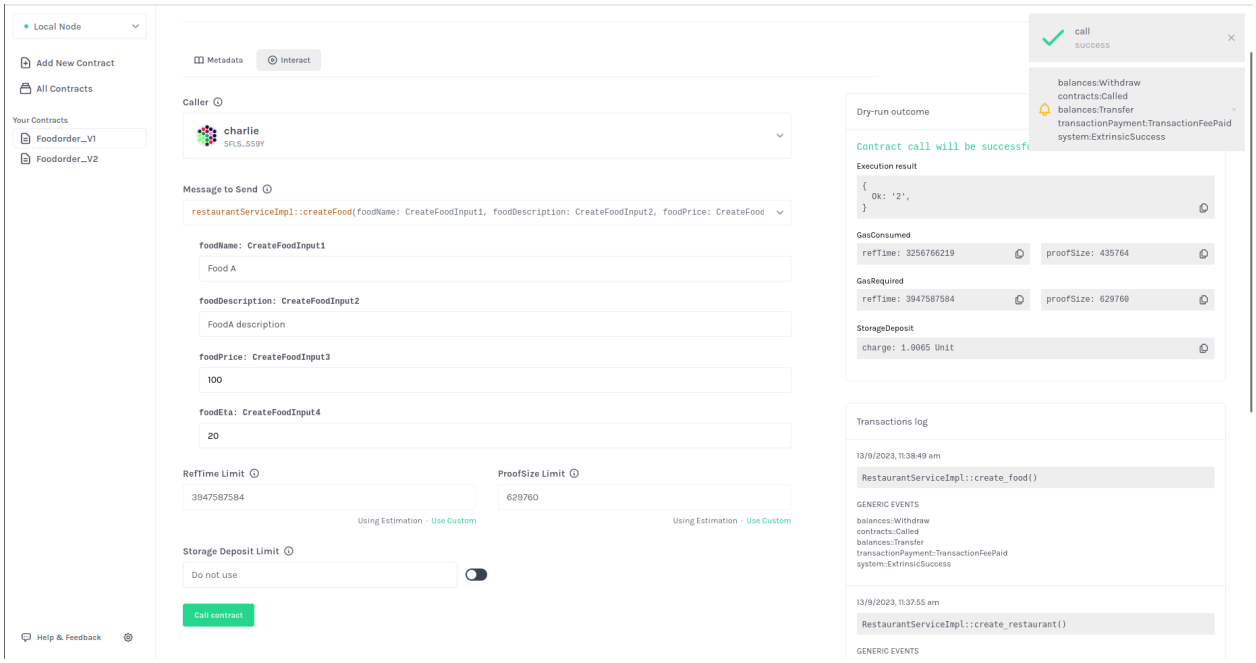
3. Create a customer in FoodOrder_V1 contract.



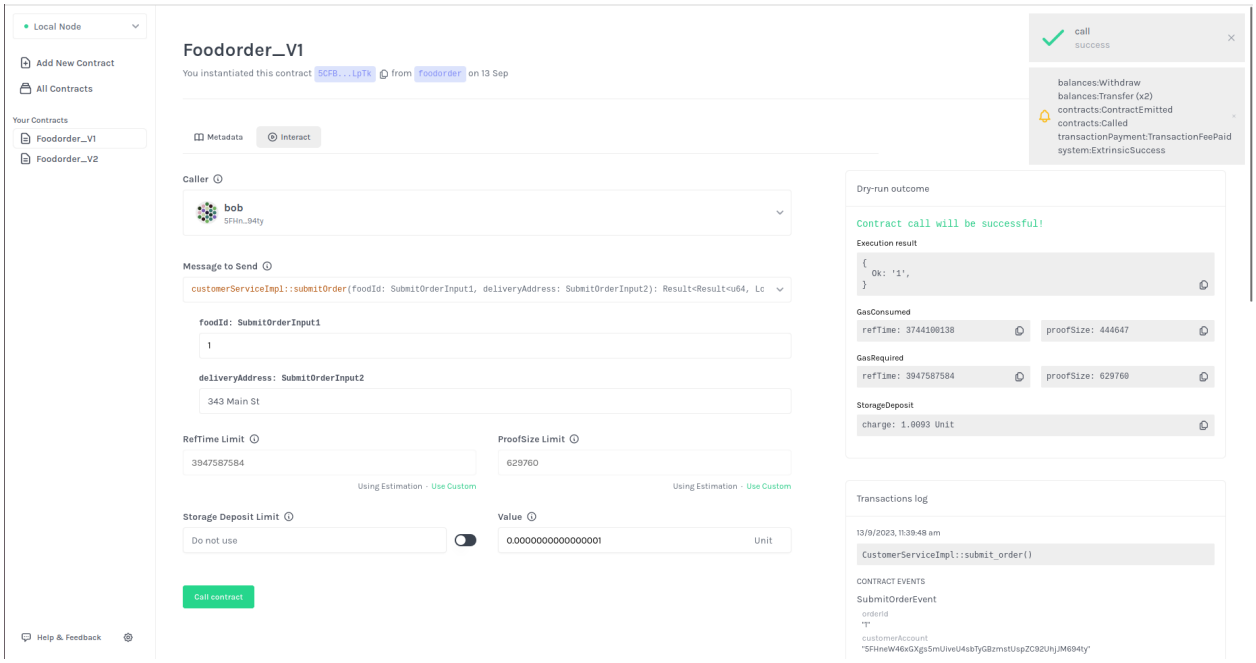
4. Create a restaurant in FoodOrder_V1 contract.



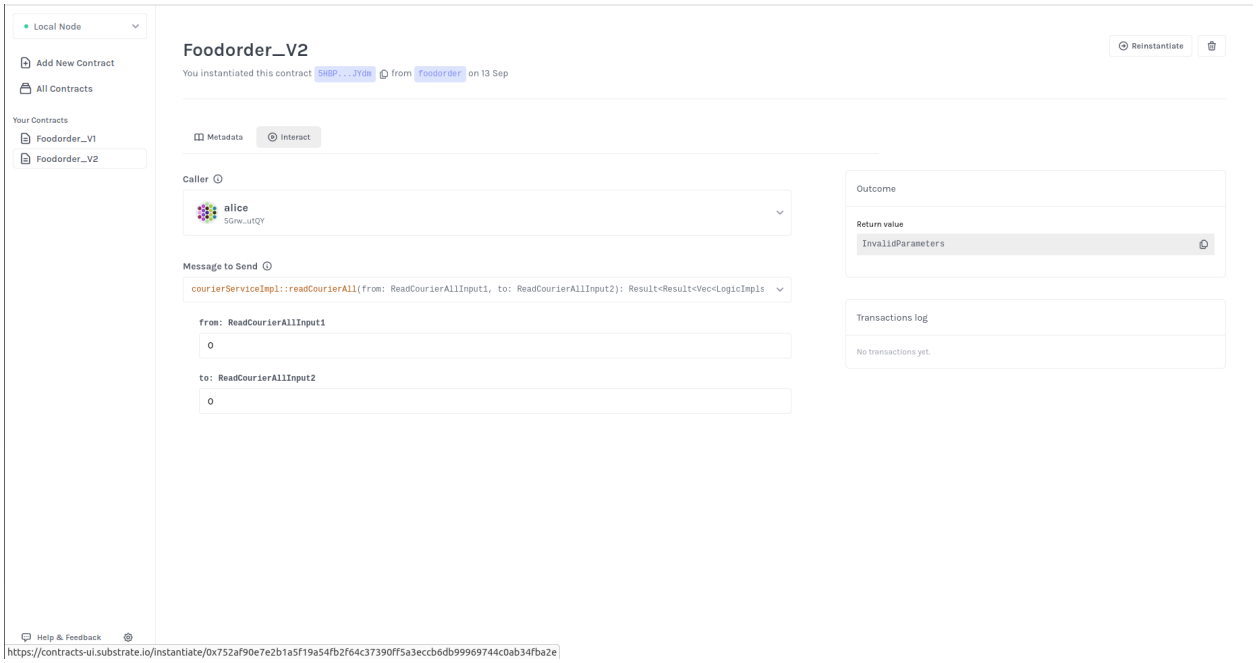
5. Create a food in FoodOrder_V1 contract.



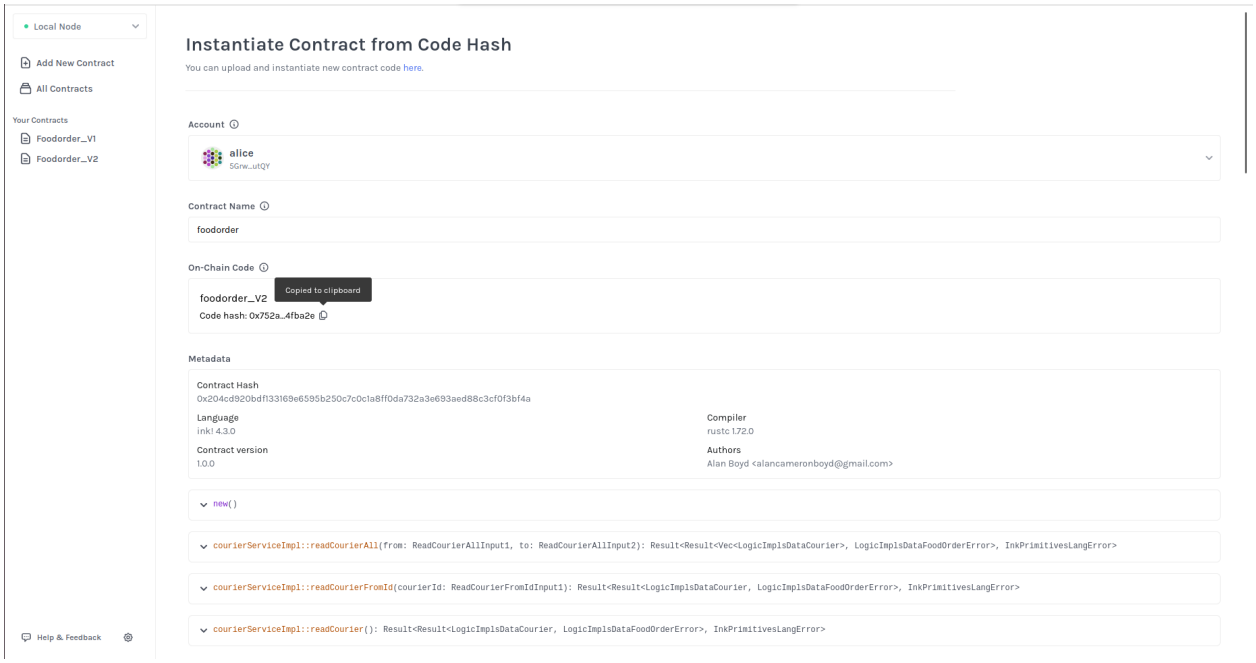
6. Submit an order in FoodOrder_V1 contract.



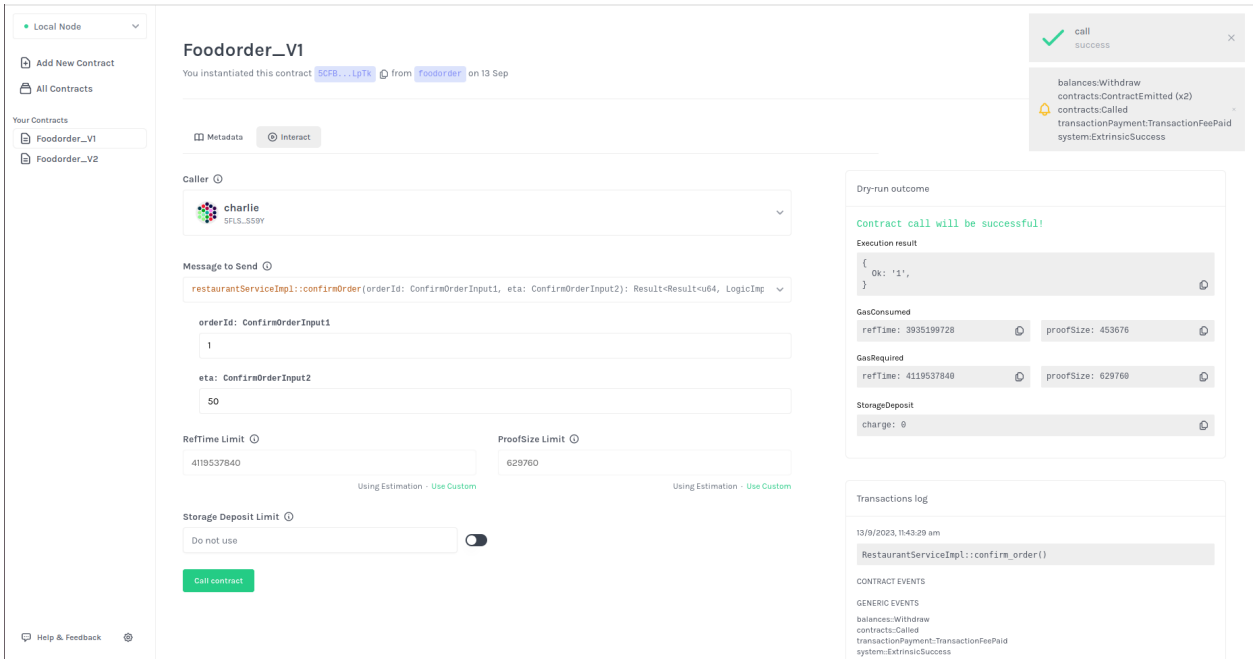
7. Instantiate FoodOrder_V2 contract.



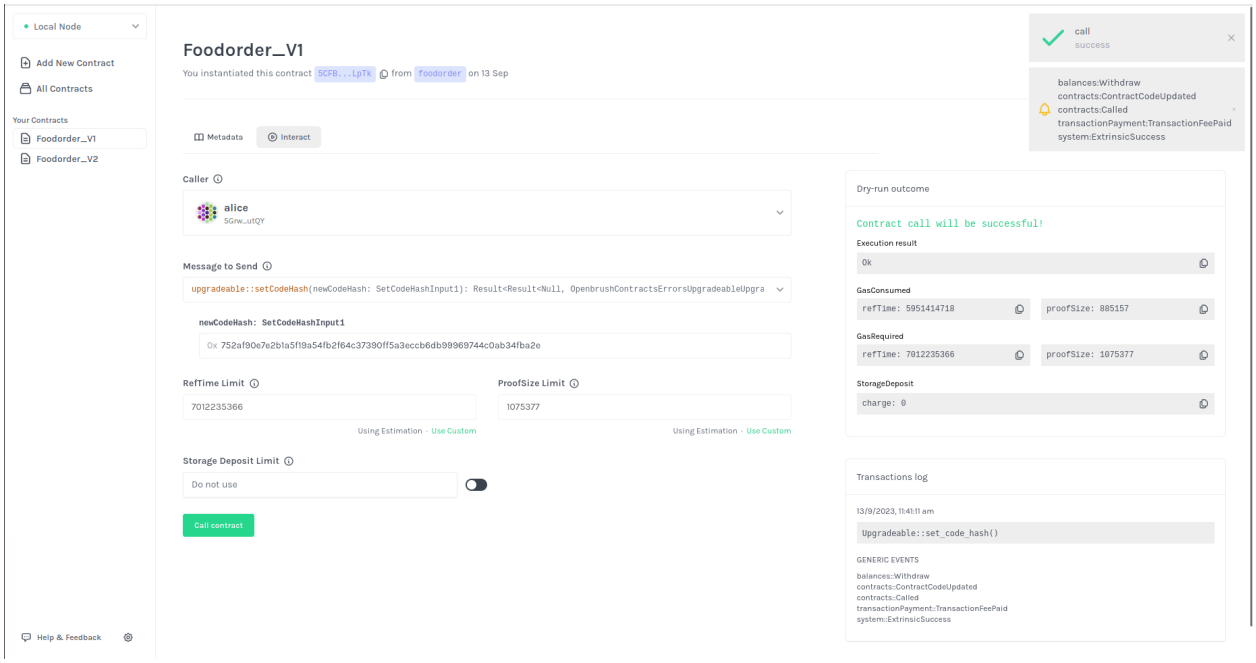
8. Copy code_hash of FoodOrder_V2 contract.



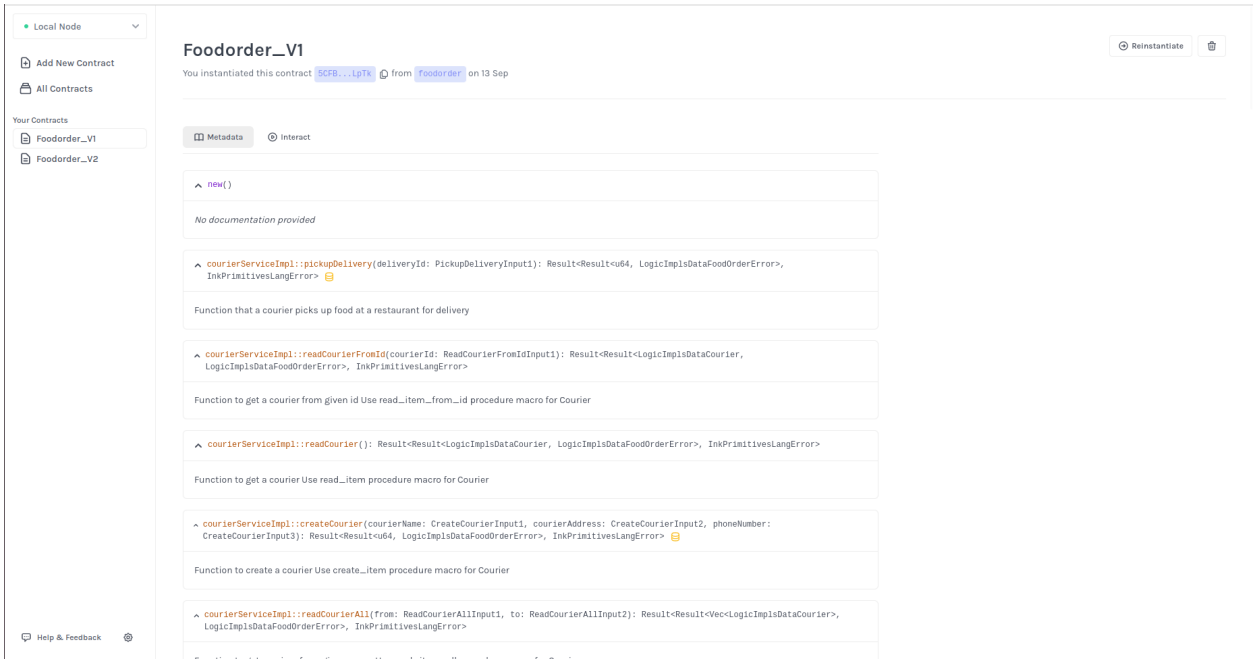
9. Confirm an order before upgrading the contract for comparison.

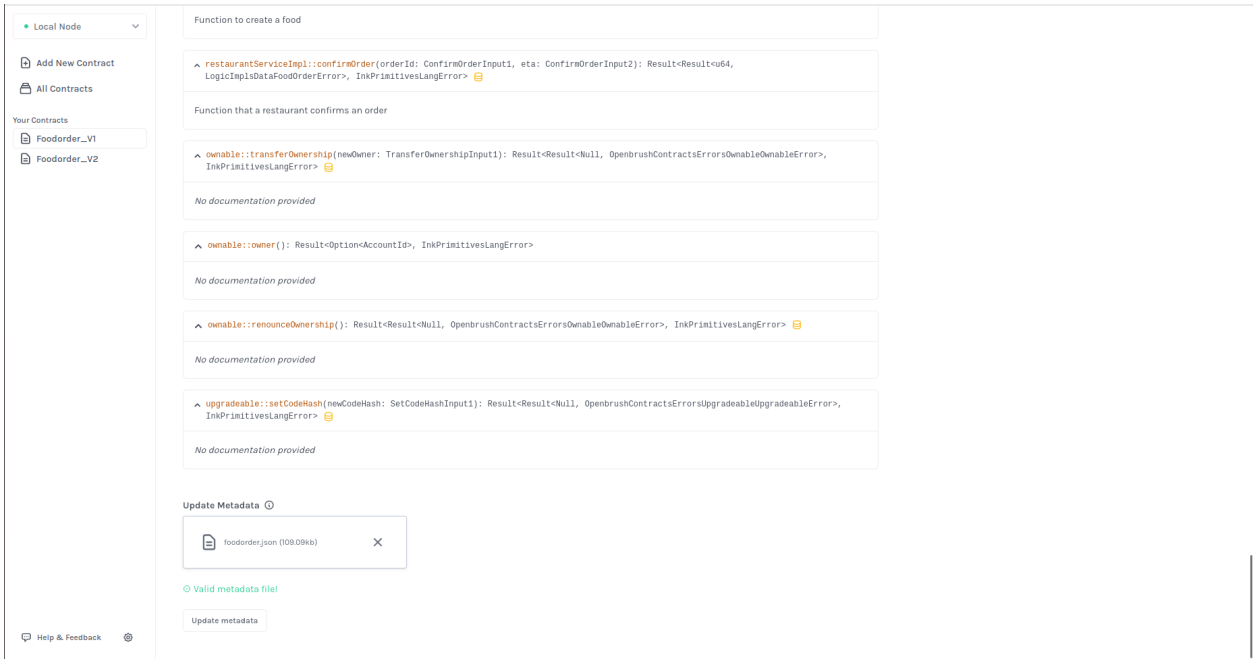


10. Set code_hash to FoodOrder_V1.

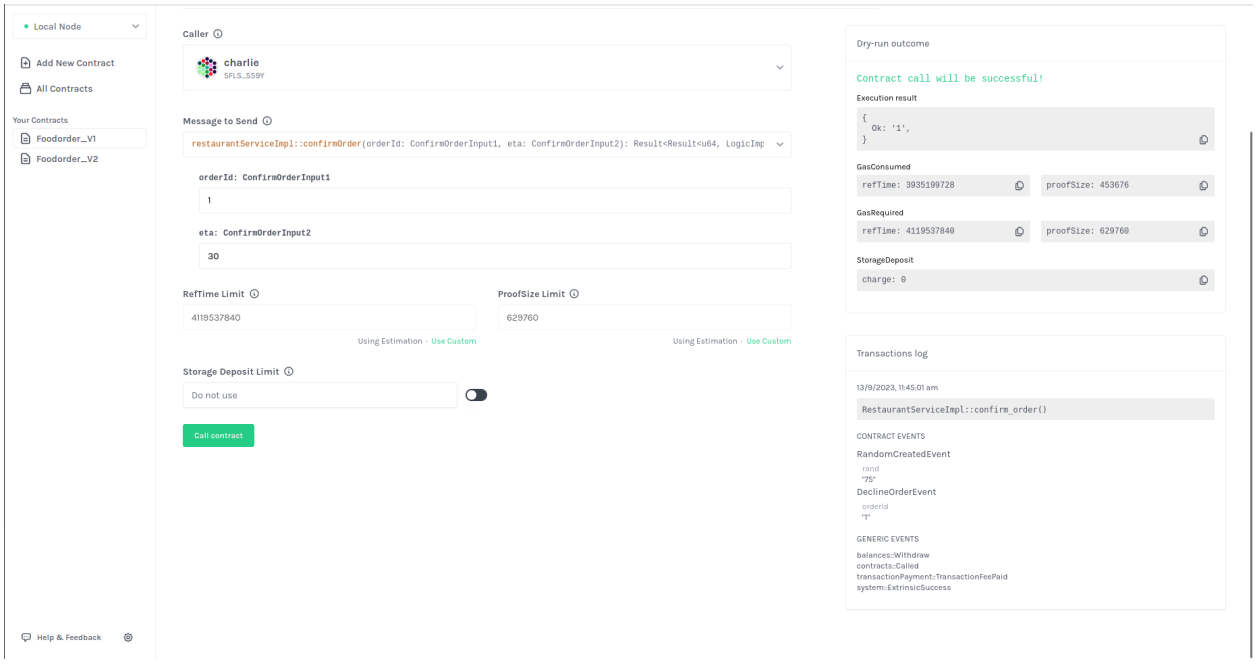


11. Update metadata to a new one.

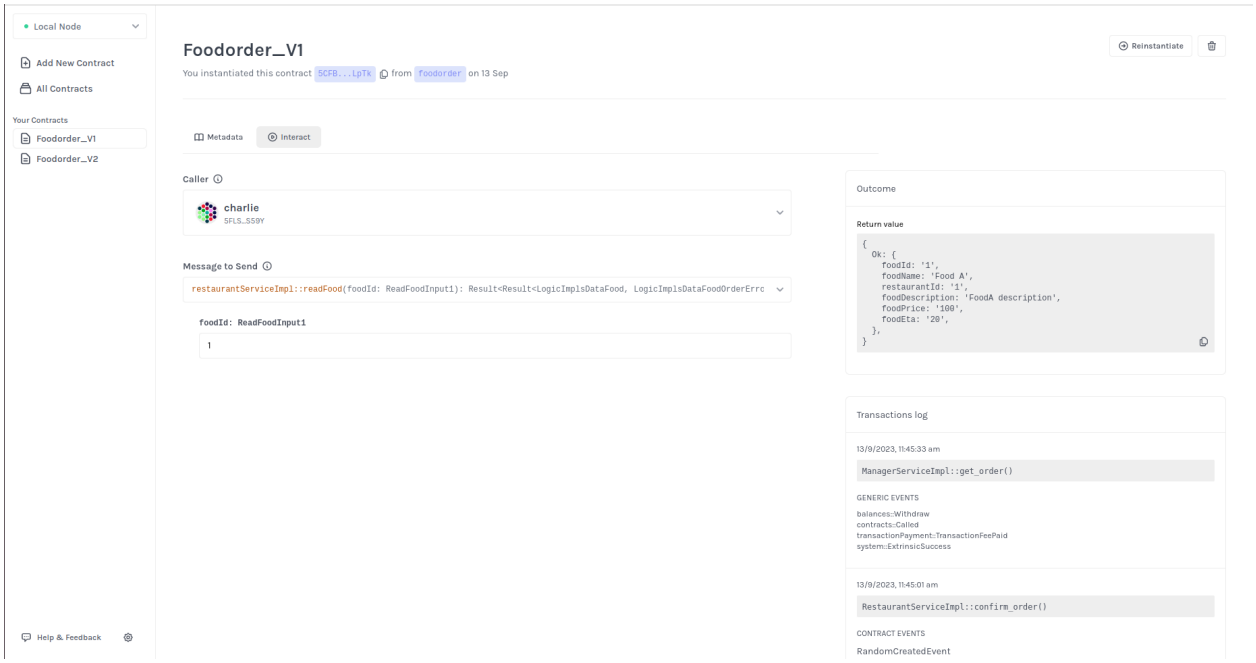




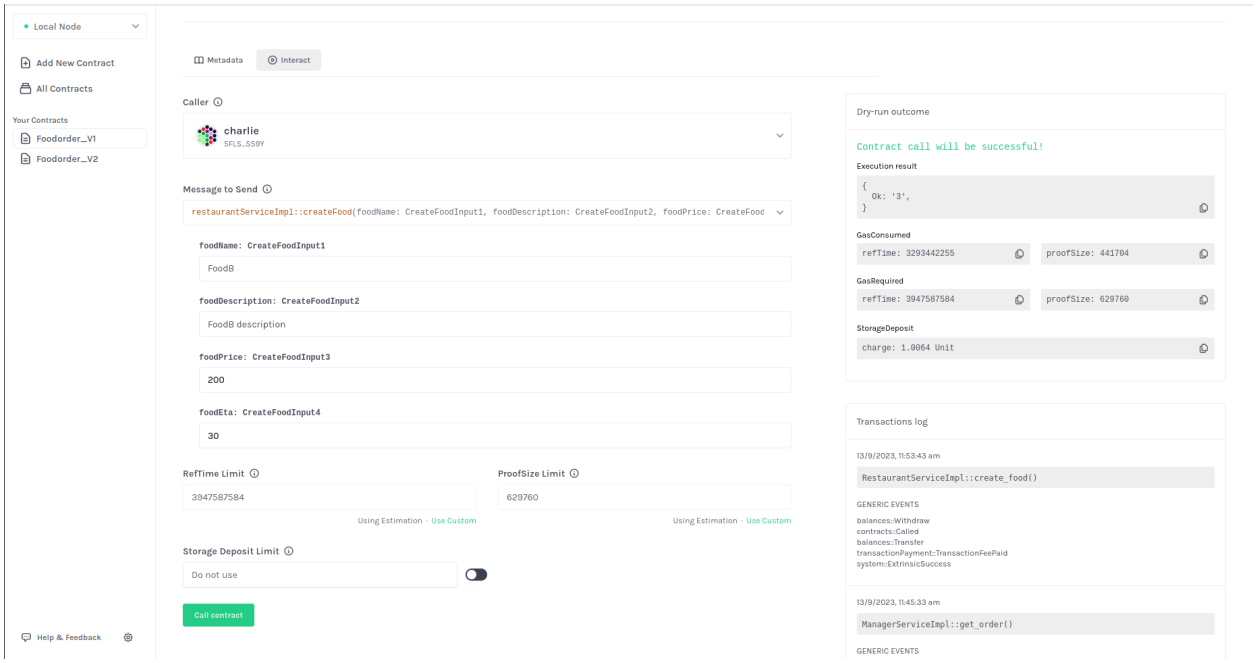
12. Confirm an order after upgrade.



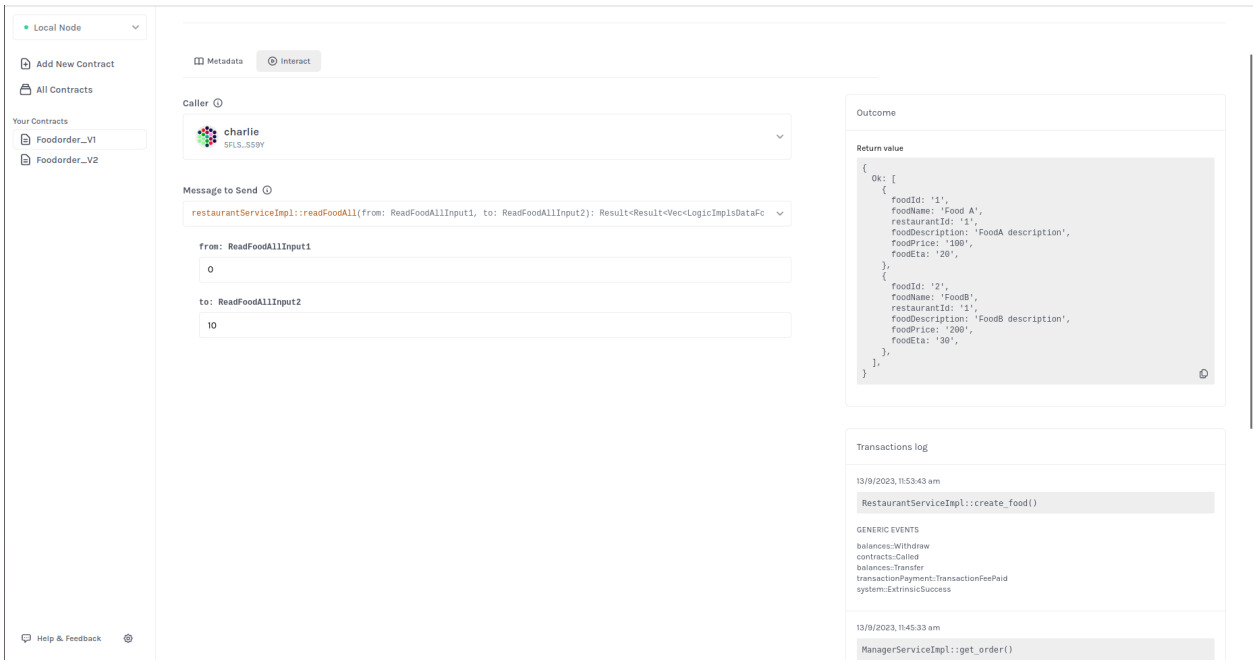
13. Read food to check the old data.



14. Create a food (this is a new data).



15. Check both old data and new data by reading all foods.



Working with Polkadot.js UI

Deployment

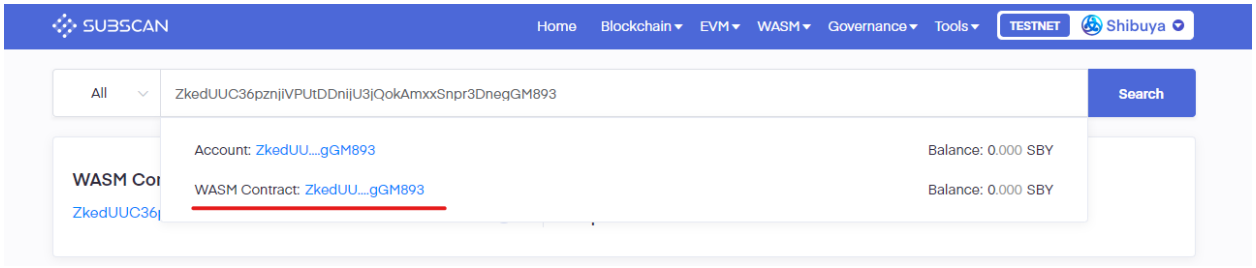
swanky contract deploy foodorder --account deploy --gas 100000 --network shibuya

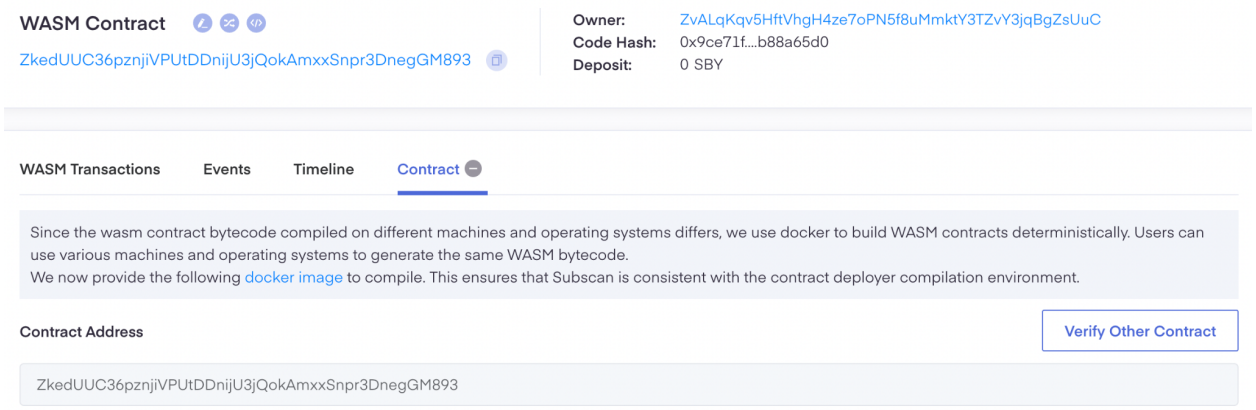
Log:

✔ Initialising OK
✔ Getting WASM OK
⁝ Connecting to node2023-08-11 11:47:18 API/INIT: shibuya/105: Not decorating runtime apis without matching versions: EthereumRuntimeRPCApi/5 (4 known)

✔ Connecting to node OK
✔ Deploying OK
✔ Writing config OK
Contract deployed!
Contract address: Yn1dHJTbKuMhA6rLLsRXQtDu4mSFGC6xtvDTueNz1axJ5Dz

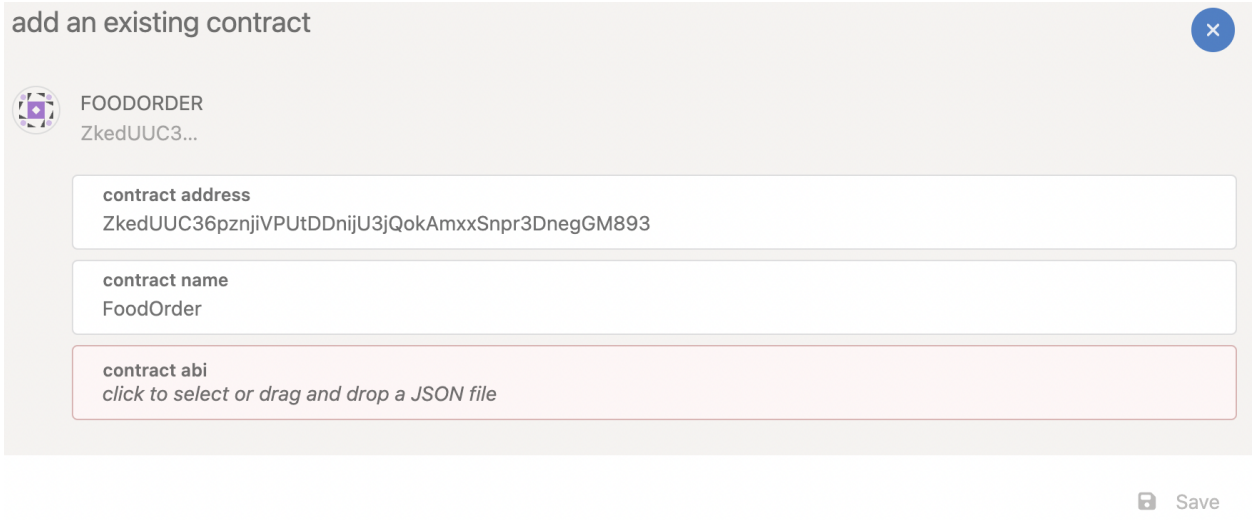
After successfully deployed, you can check the deployed contract on the shibuya blockexplorer https://shibuya.subscan.io/.



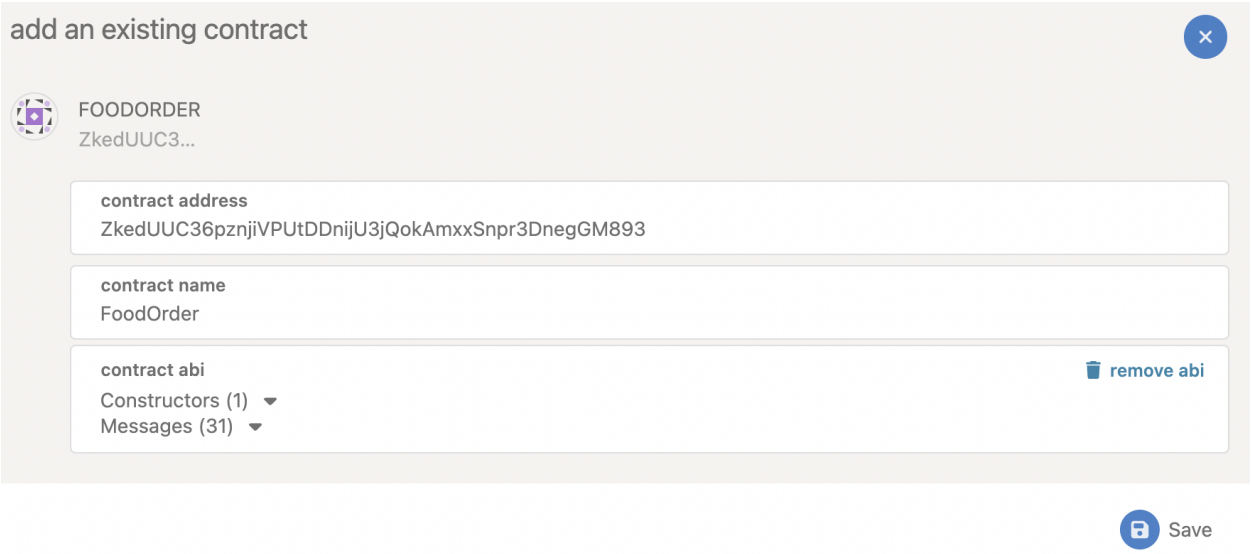


Interacting with the smart contract

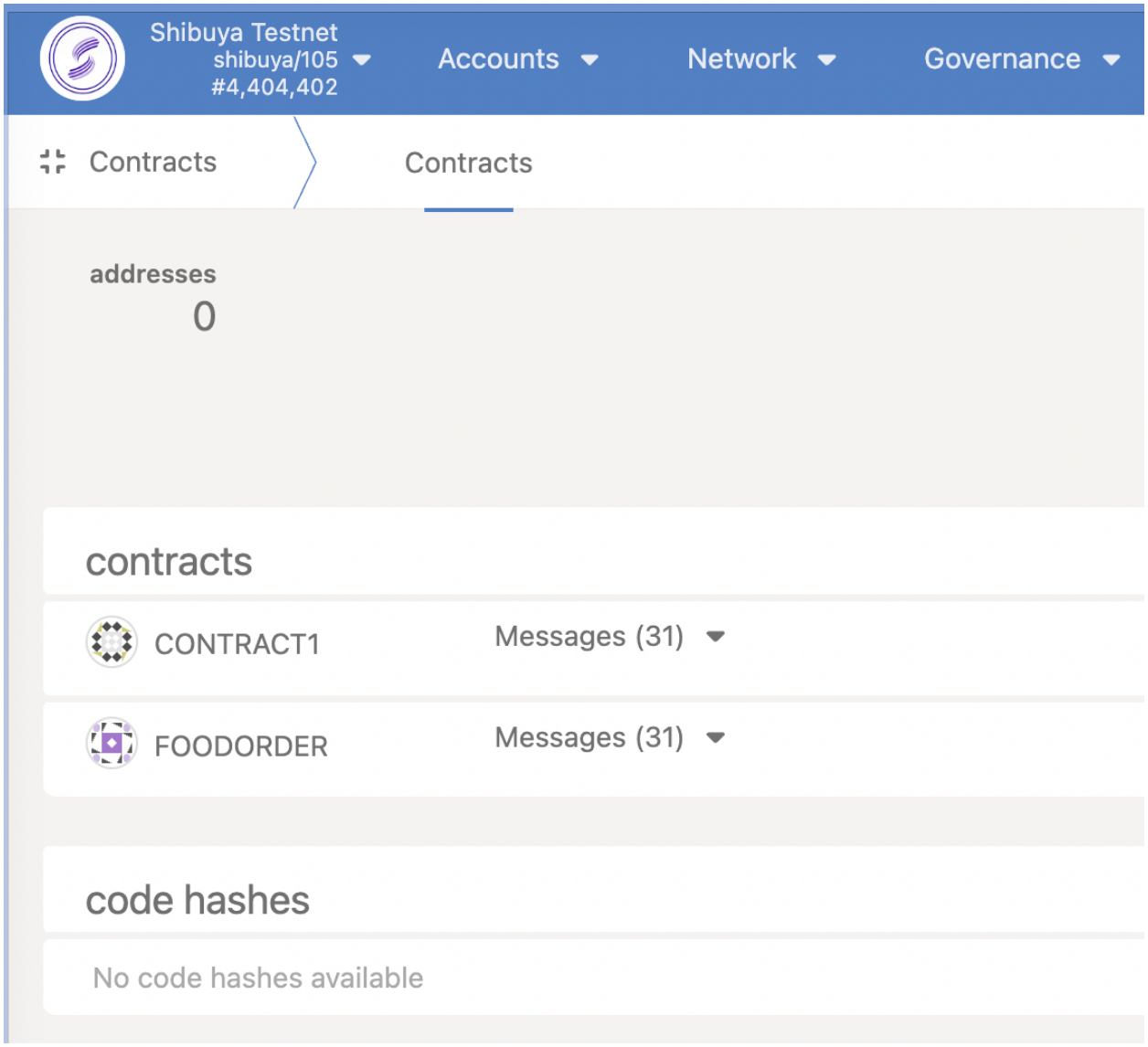
Click the button “Add an existing contract”



Type the contract address to the relevant input and upload the ABI JSON file in the contract abi box



After that, you can see a new smart contract named FOODORDER



Here, you can see a list of all messages of the smart contract, some read(able) some exec(cutable). Now we can manually simulate a food order flow in steps.

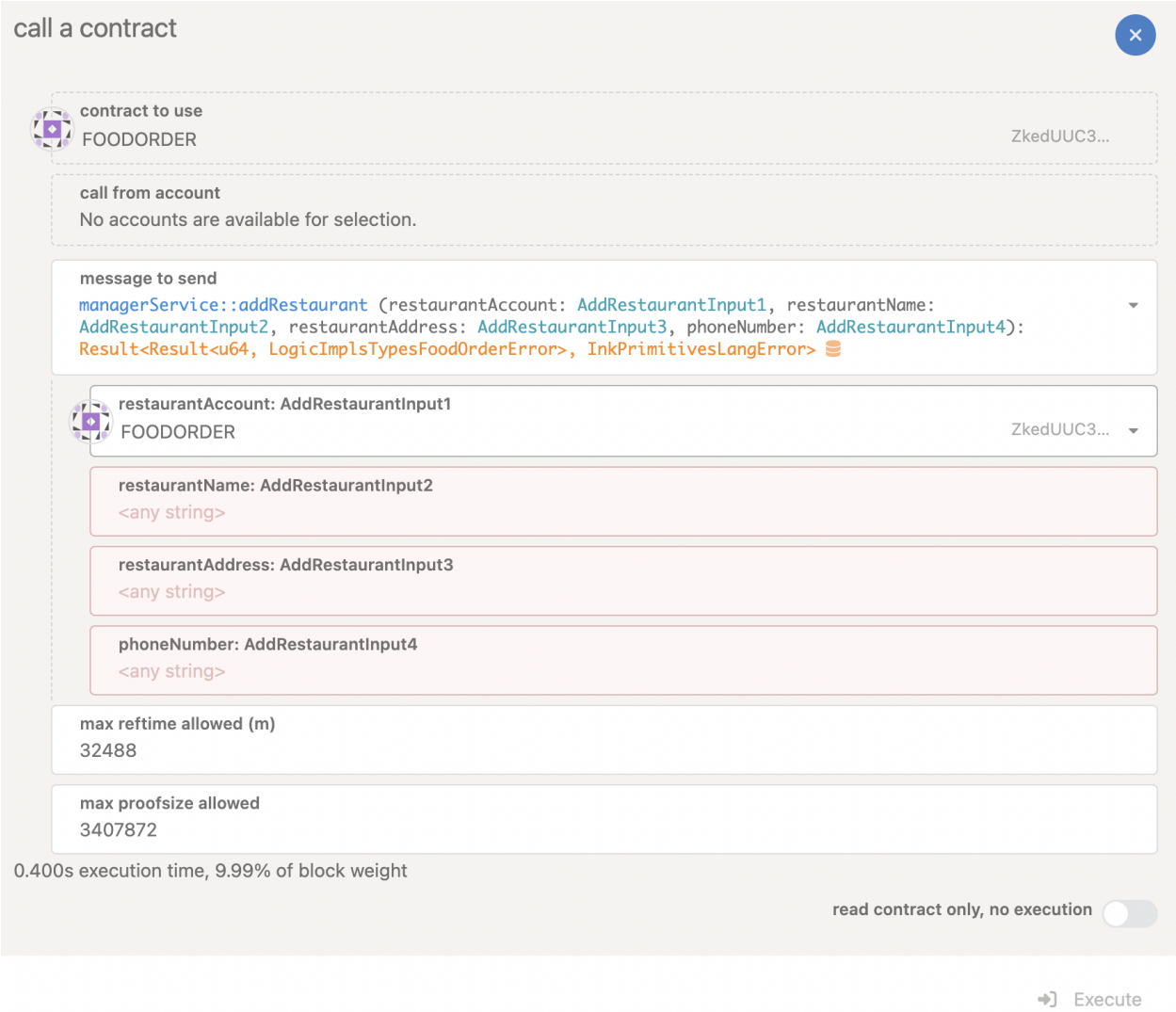
contracts

CONTRACT1 — Messages (31)

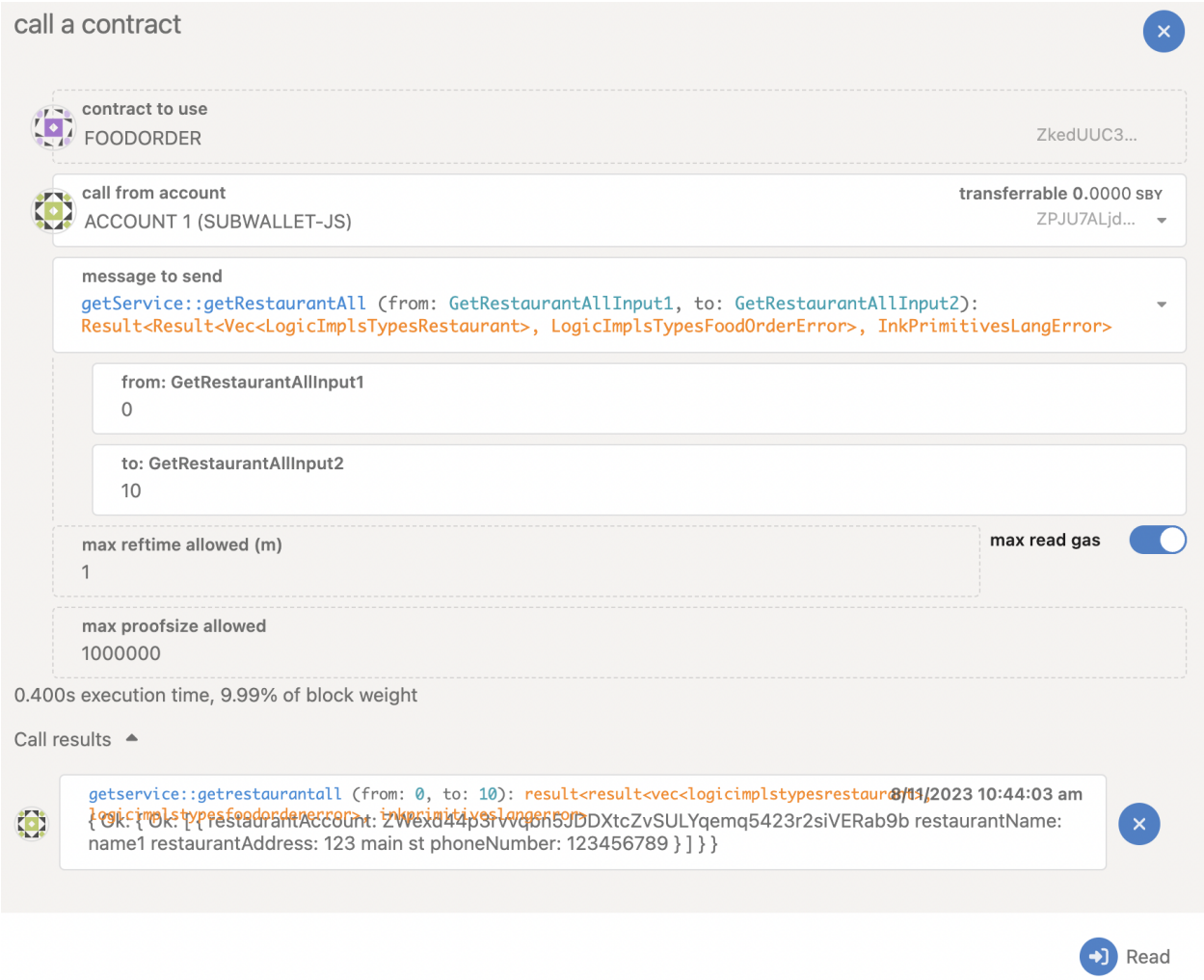
FOODORDER — Messages (31)

- exec — `courierService::pickupDelivery (deliveryId: PickupDeliveryInput1): Result<Result<u64, LogicImplsTypesFoodOrderError>, InkPrimitivesLangError>`
  No documentation provided
- exec — `customerService::acceptDelivery (deliveryId: AcceptDeliveryInput1): Result<Result<u64, LogicImplsTypesFoodOrderError>, InkPrimitivesLangError>`
  No documentation provided
- exec — `customerService::addCustomer (customerName: AddCustomerInput1, customerAddress: AddCustomerInput2, phoneNumber: AddCustomerInput3): Result<Result<u64, LogicImplsTypesFoodOrderError>, InkPrimitivesLangError>`
  No documentation provided
- exec — `customerService::submitOrder (foodId: SubmitOrderInput1, deliveryAddress: SubmitOrderInput2): Result<Result<u64, LogicImplsTypesFoodOrderError>, InkPrimitivesLangError>`
  No documentation provided
- exec — `managerService::addCourier (courierAccount: AddCourierInput1, courierName: AddCourierInput2, courierAddress: AddCourierInput3, phoneNumber: AddCourierInput4): Result<Result<u64, LogicImplsTypesFoodOrderError>, InkPrimitivesLangError>`
  No documentation provided
- exec — `managerService::addRestaurant (restaurantAccount: AddRestaurantInput1, restaurantName: AddRestaurantInput2, restaurantAddress: AddRestaurantInput3, phoneNumber: AddRestaurantInput4): Result<Result<u64, LogicImplsTypesFoodOrderError>, InkPrimitivesLangError>`
  No documentation provided
- exec — `managerService::changeFeeRate (rate: ChangeFeeRateInput1): Result<Result<Text, LogicImplsTypesFoodOrderError>, InkPrimitivesLangError>`
  No documentation provided
- exec — `managerService::changeManager (newAccount: ChangeManagerInput1): Result<Result<Text, LogicImplsTypesFoodOrderError>, InkPrimitivesLangError>`
  No documentation provided
- exec — `restaurantService::addFood (foodName: AddFoodInput1, description: AddFoodInput2, price: AddFoodInput3, eta: AddFoodInput4): Result<Result<u64, LogicImplsTypesFoodOrderError>, InkPrimitivesLangError>`
  No documentation provided

1. Add a restaurant.

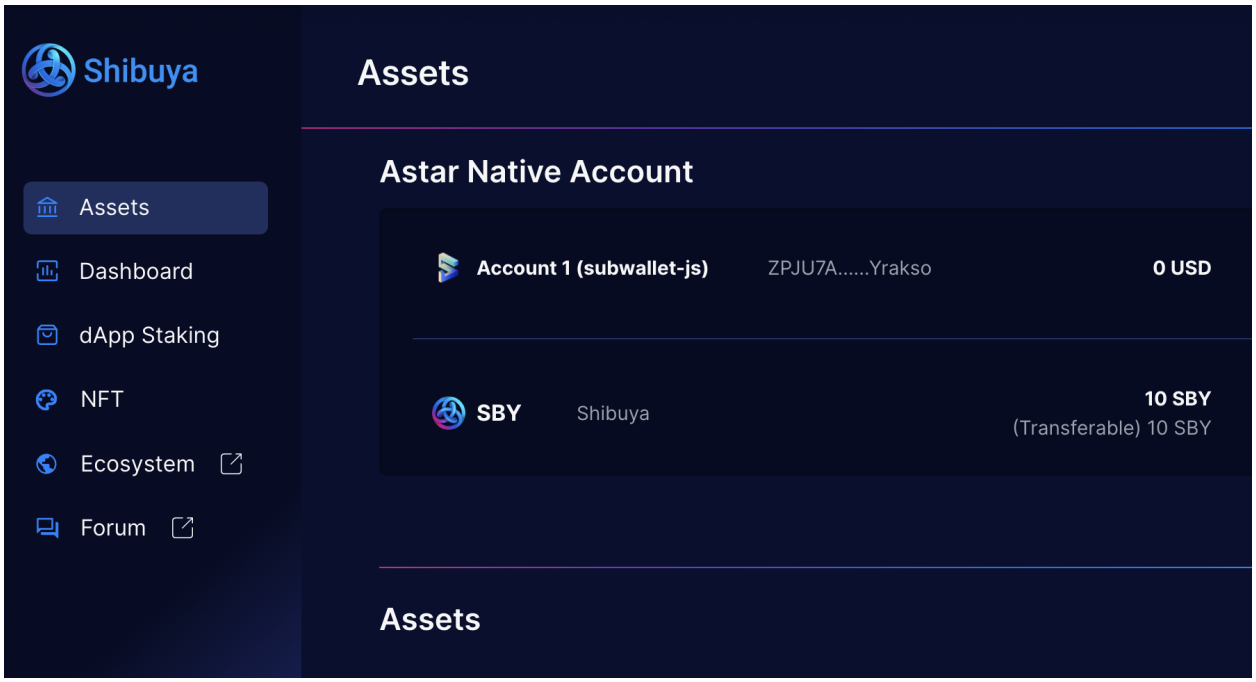


After you added a restaurant, you can check the result here by invoking the getRestaurantAll message. Here, you can set from (0) and to (e.g. 10) as parameters to paginate through existing restaurants listing in contract storage:



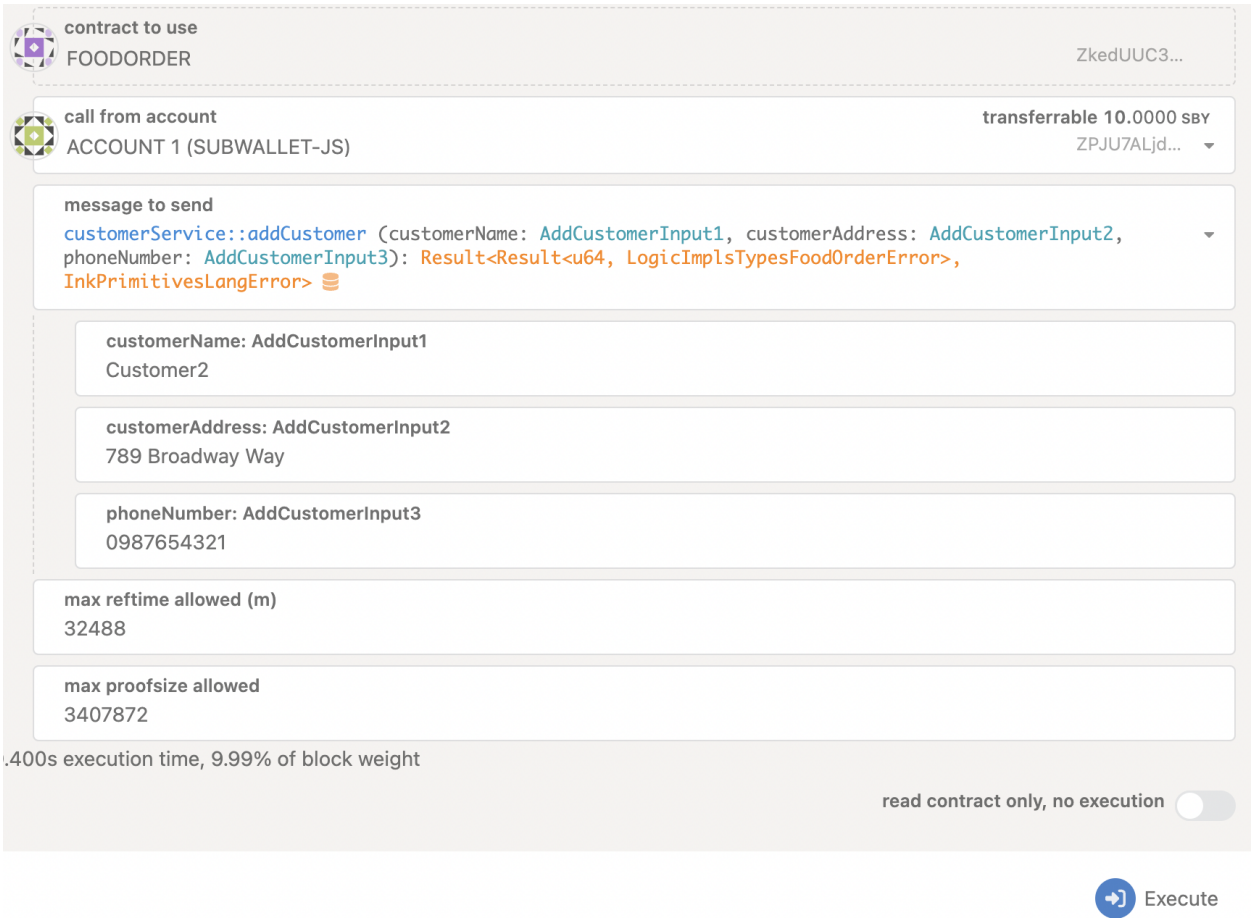
2. Get free SBY tokens from a faucet.

Before you are trying to execute a transaction, you will need to get native tokens. You can get faucet from here.

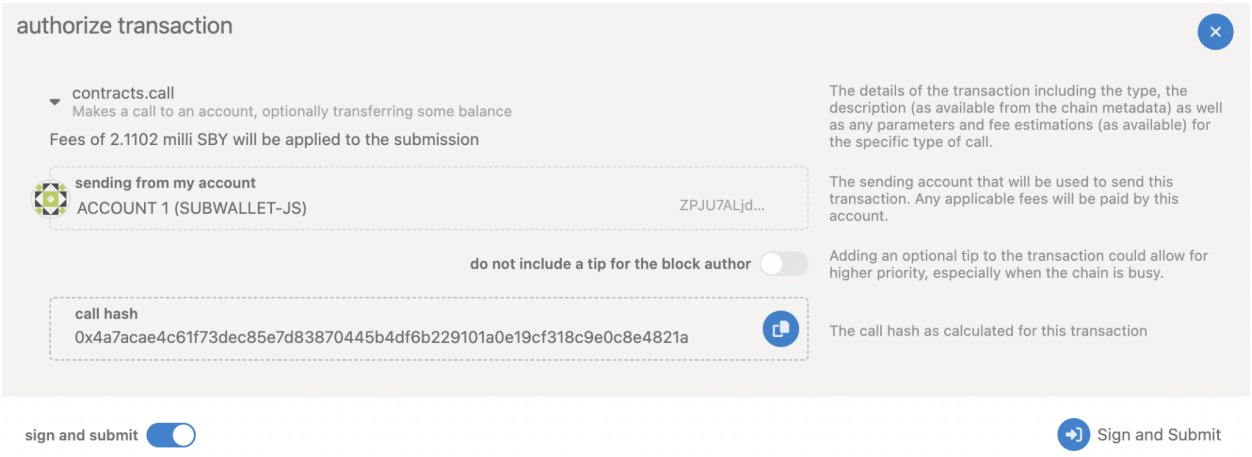


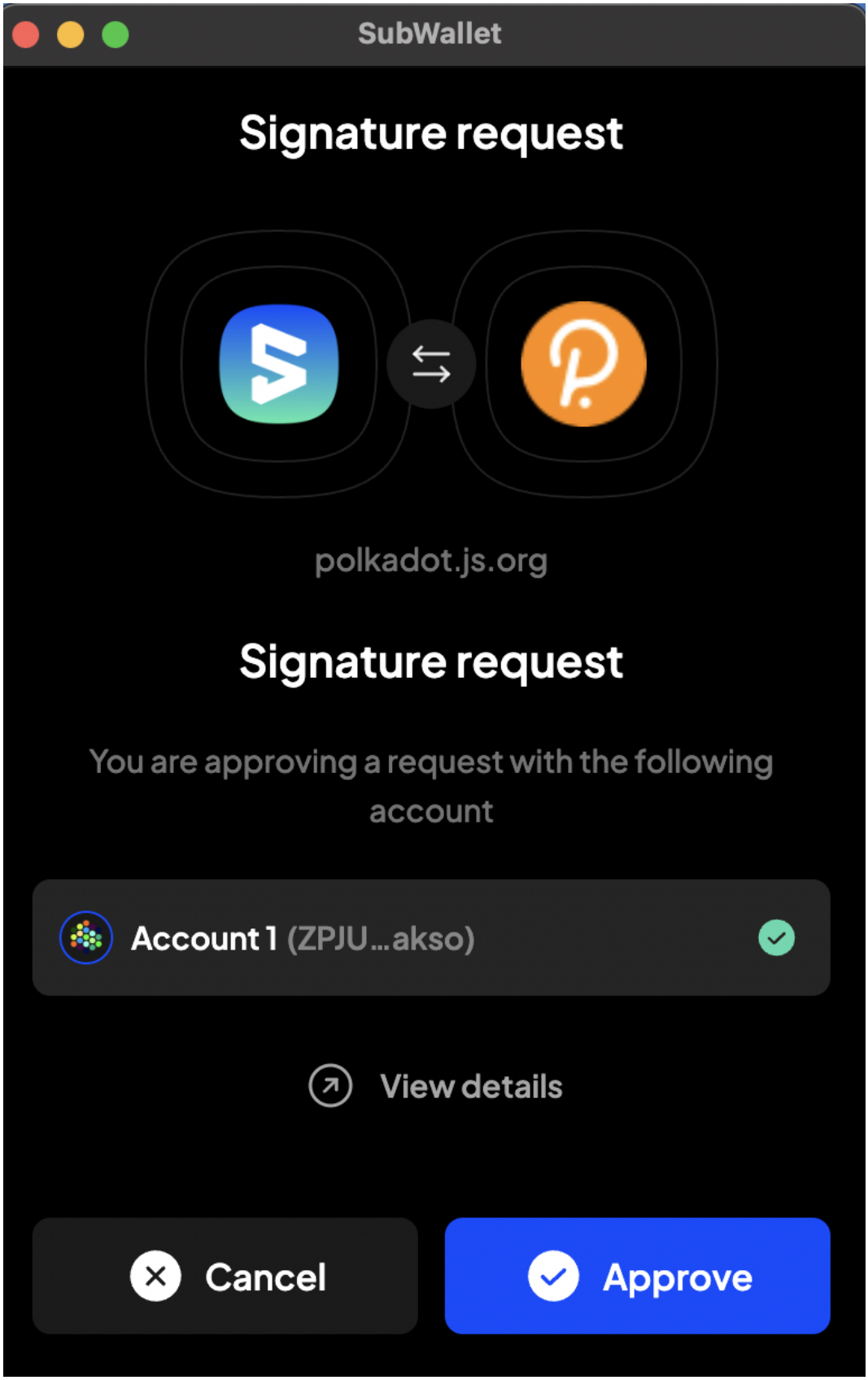
If it succeeds, then you can see such a result.

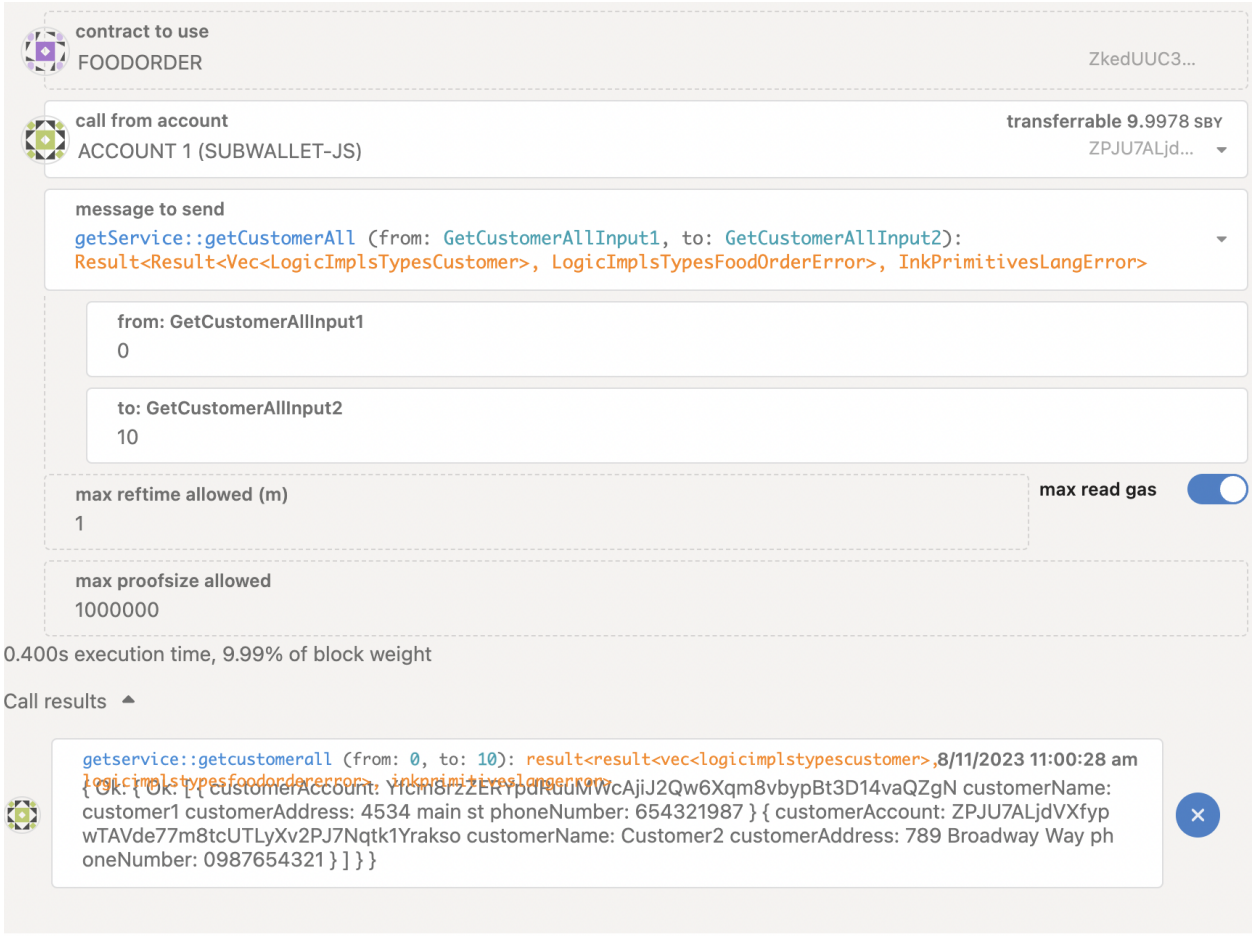
3. Add a customer.



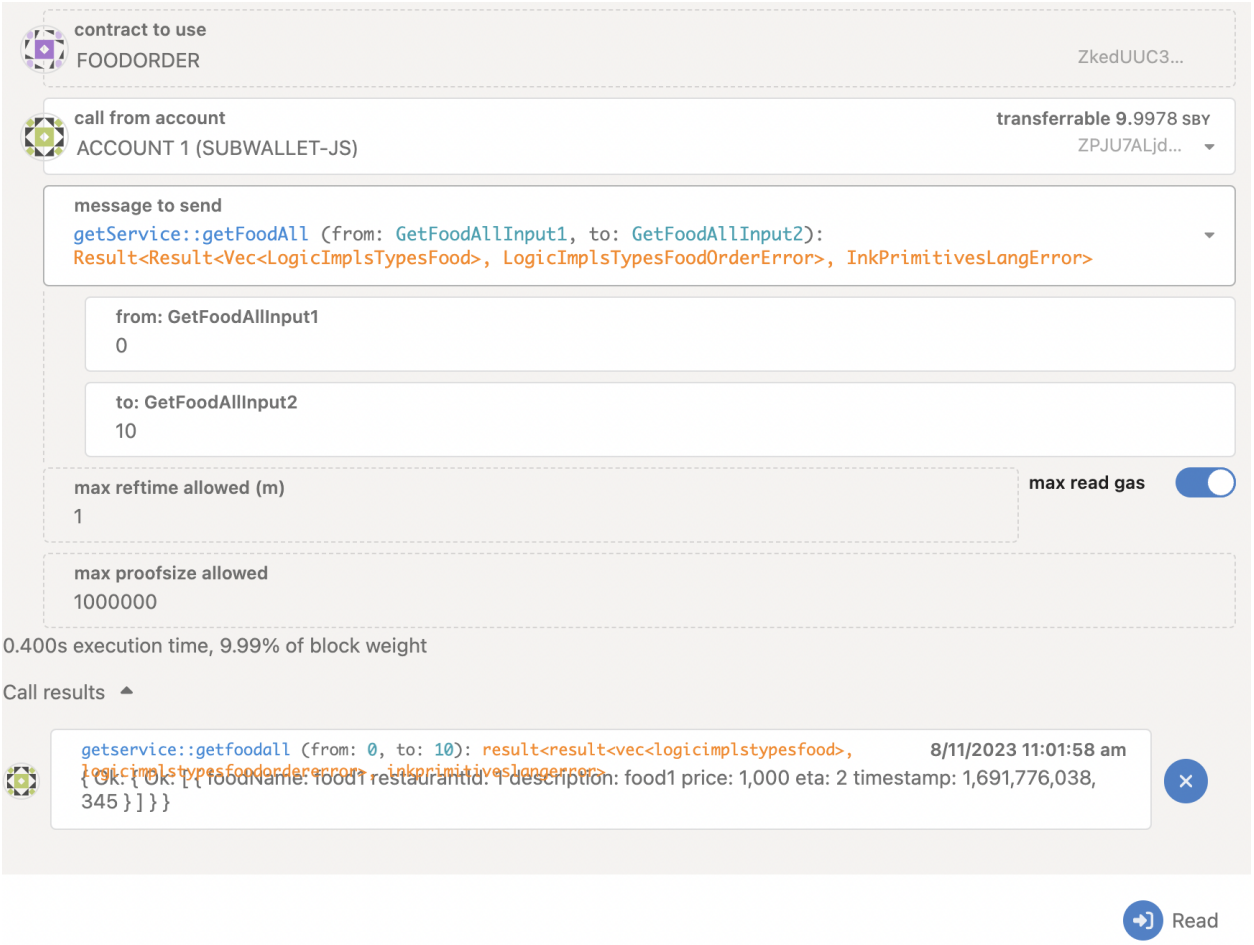
After you complete the form, then click the “Sign and Submit” button from your Sub wallet browser popup to pay the gas fees.



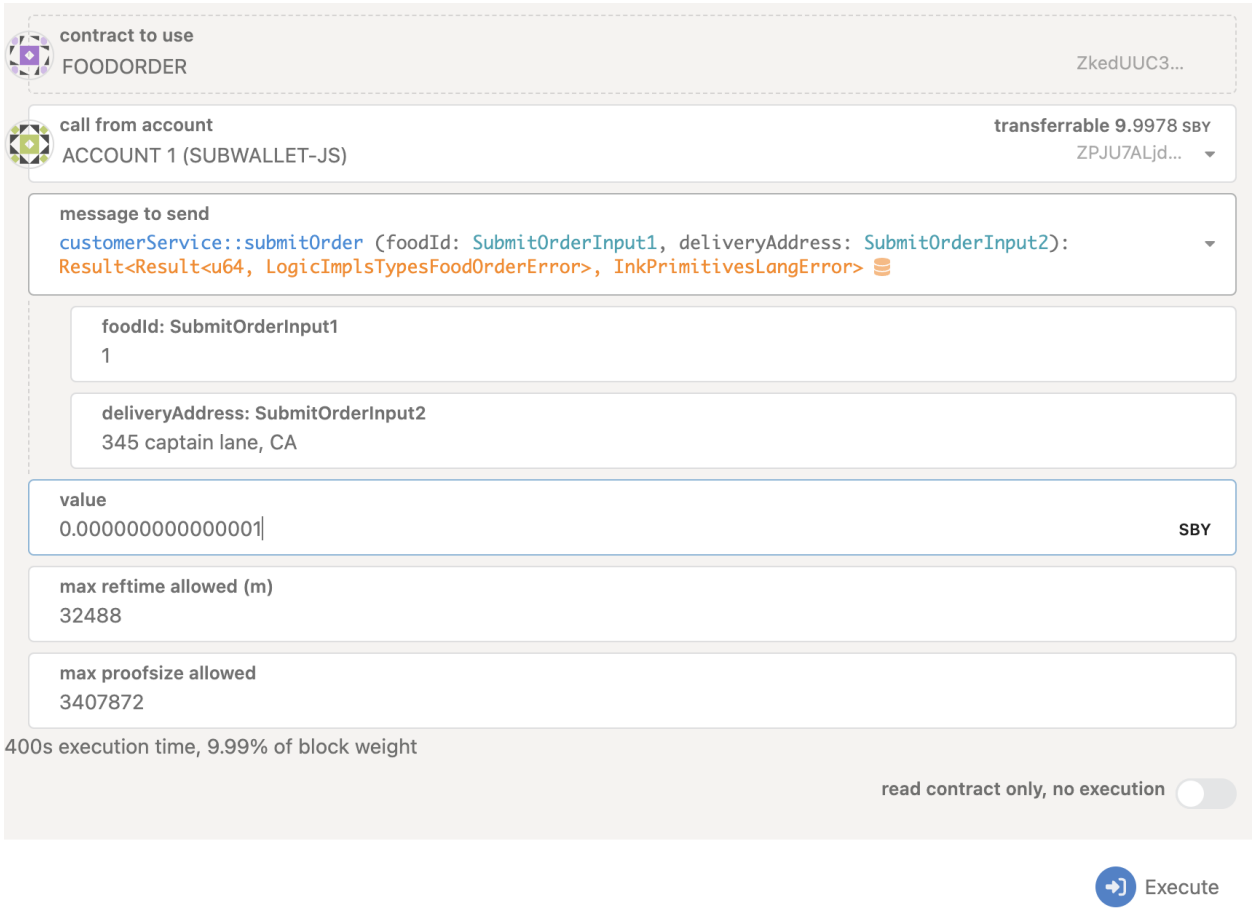

SubWallet
Signature request
polkadot.js.org
Signature request
You are approving a request with the following account
Account 1 (ZPJU...akso)
View details
Cancel
Approve

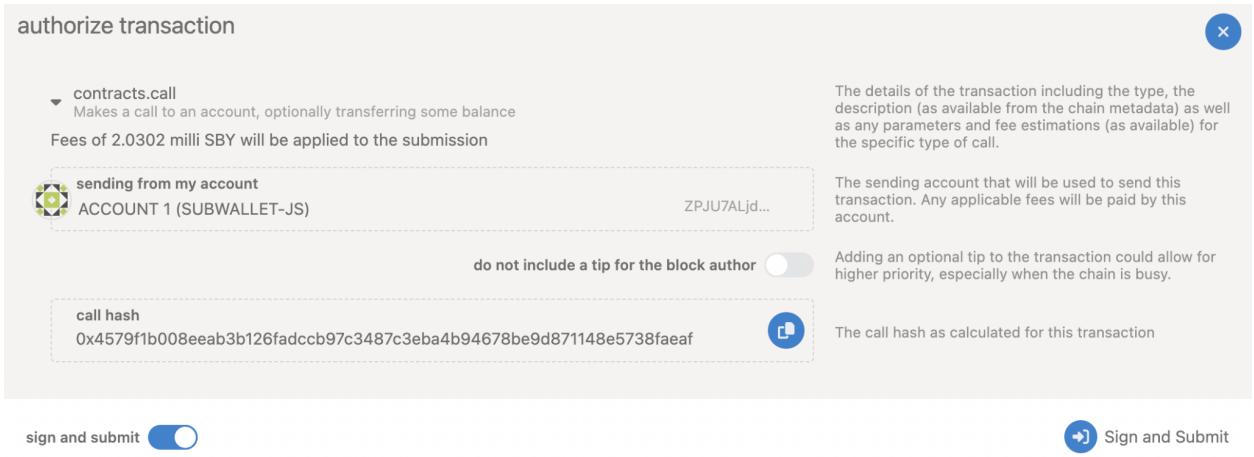

contract to use
FOODORDER
ZkedUUC3...
call from account
ACCOUNT 1 (SUBWALLET-JS)
transferrable 9.9978 SBY
ZPJU7ALjd...
message to send
getService::getCustomerAll (from: GetCustomerAllInput1, to: GetCustomerAllInput2):
Result<Result<Vec<LogicImplsTypesCustomer>, LogicImplsTypesFoodOrderError>, InkPrimitivesLangError>
from: GetCustomerAllInput1
0
to: GetCustomerAllInput2
10
max reftime allowed (m)
1
max read gas
max proofsize allowed
1000000
0.400s execution time, 9.99% of block weight
Call results
getservice::getcustomerall (from: 0, to: 10): result<result<vec<logicimplstypescustomer>,8/11/2023 11:00:28 am
{ Ok: { Ok: [{ customerAccount: YfcM8rZZERYpdRuuMWcAjiJ2Qw6Xqm8vbypBt3D14vaQZgN customerName:
customer1 customerAddress: 4534 main st phoneNumber: 654321987 } { customerAccount: ZPJU7ALjdVXfyp
wTAVde77m8tcUTLyXv2PJ7Nqtk1Yrakso customerName: Customer2 customerAddress: 789 Broadway Way ph
oneNumber: 0987654321 }] } }
Read

4. List food items

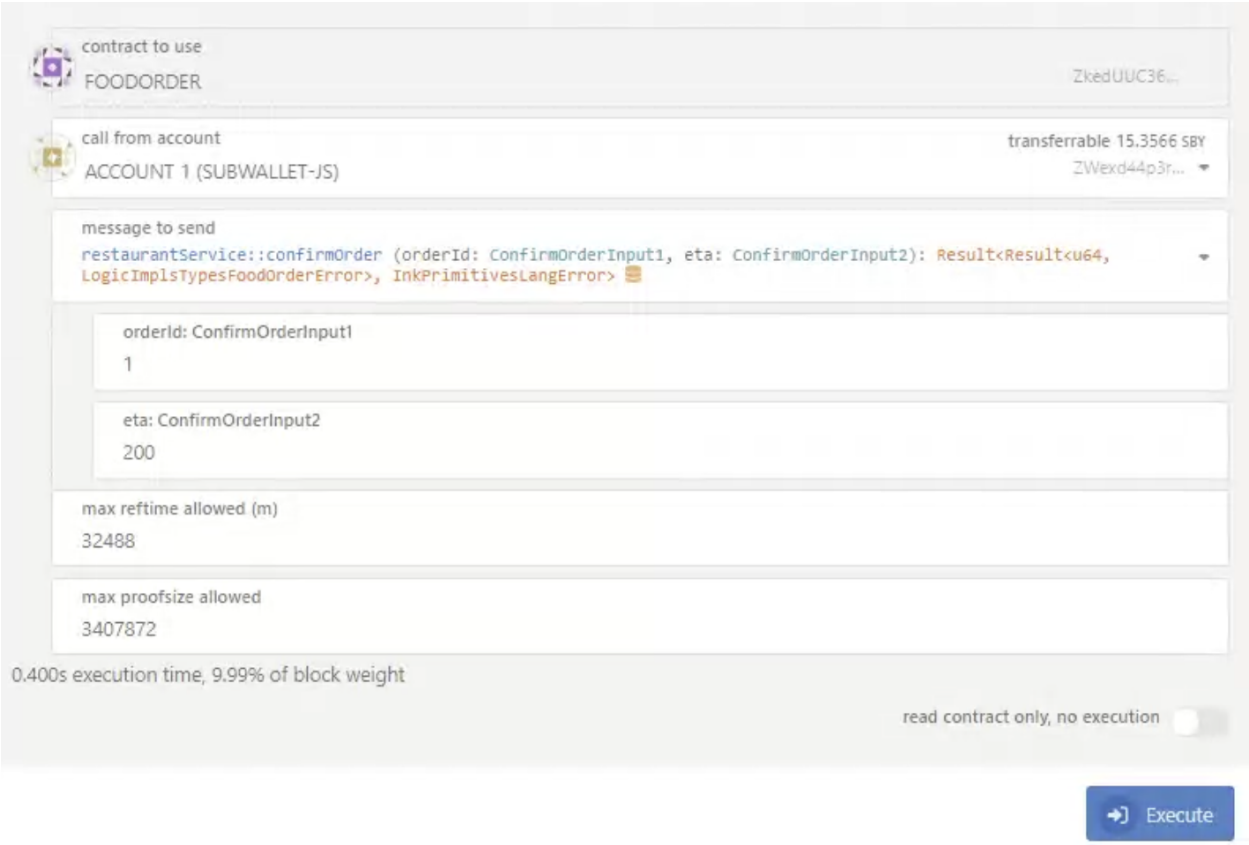


5. Consumer submits a food order

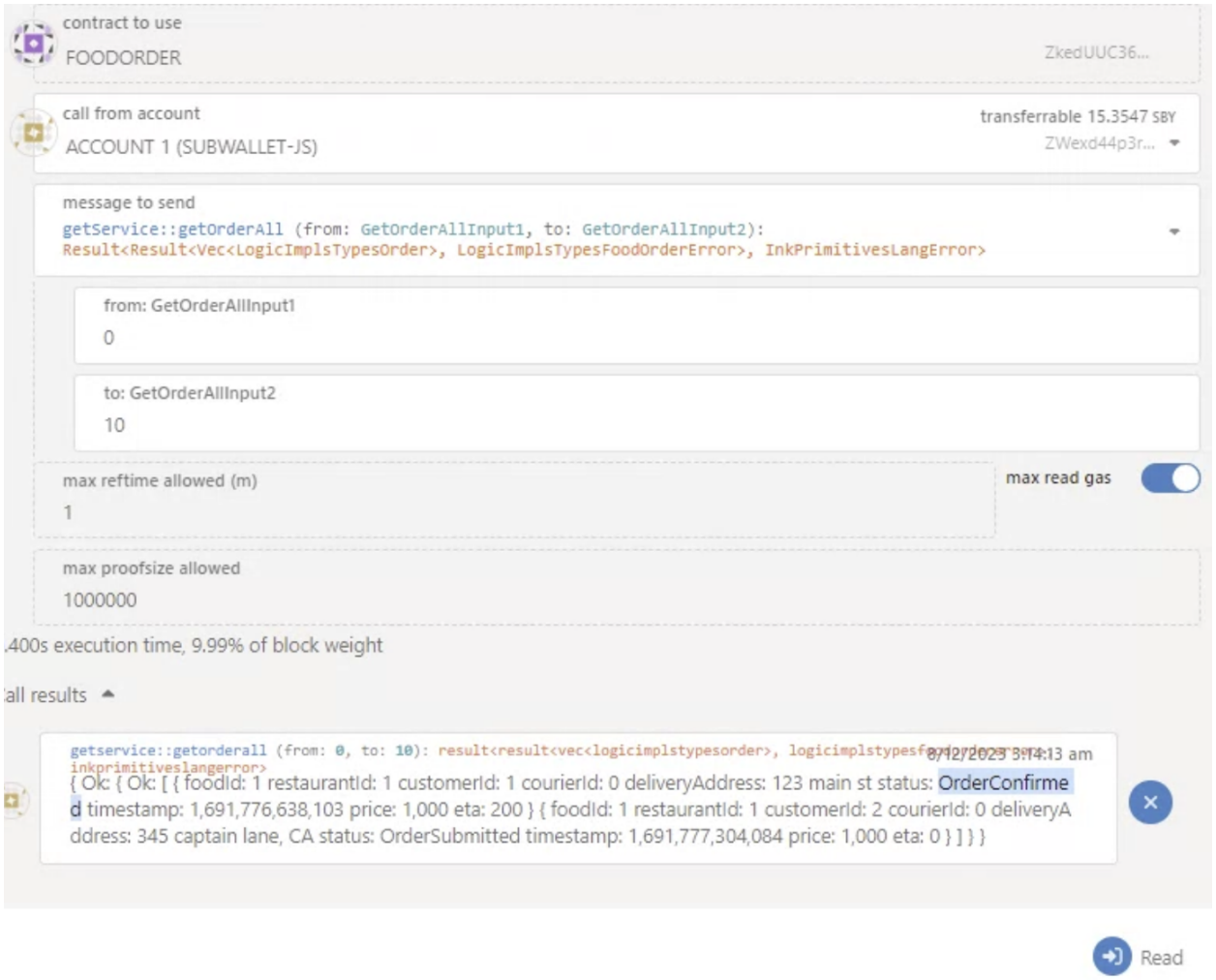




6. Confirm a order

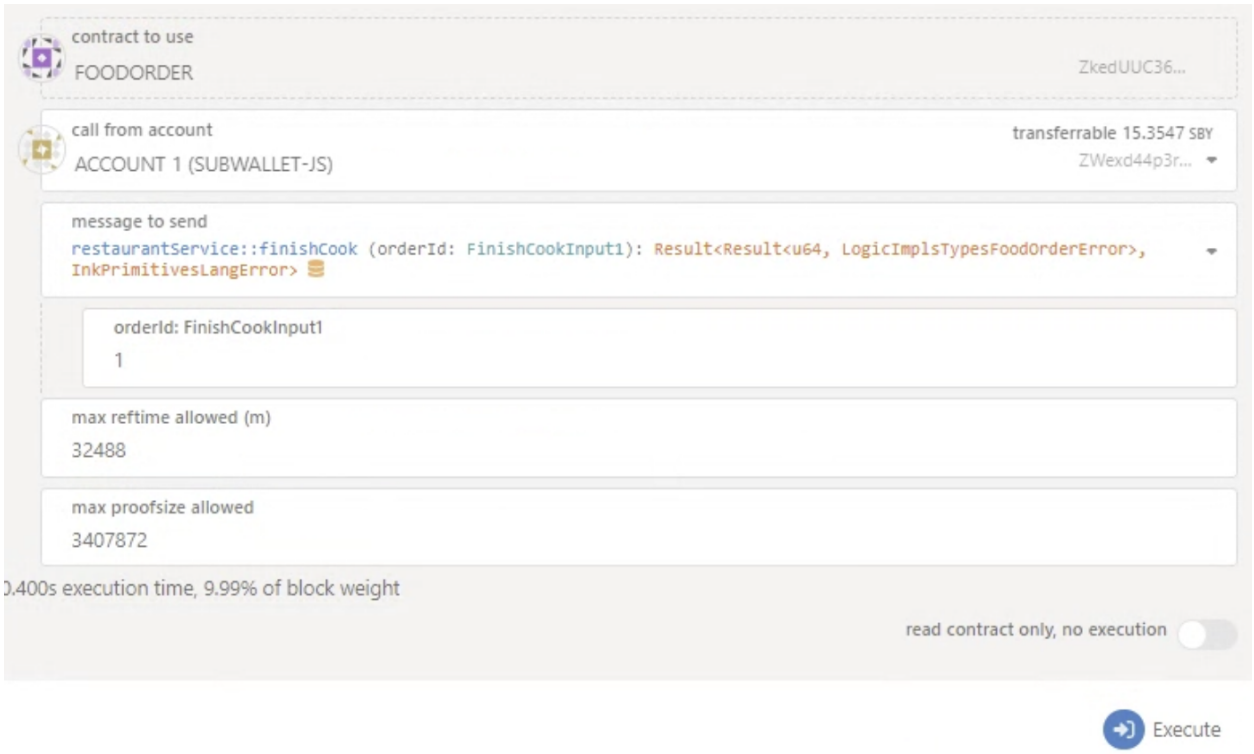


After you confirm an order, let's check its status.

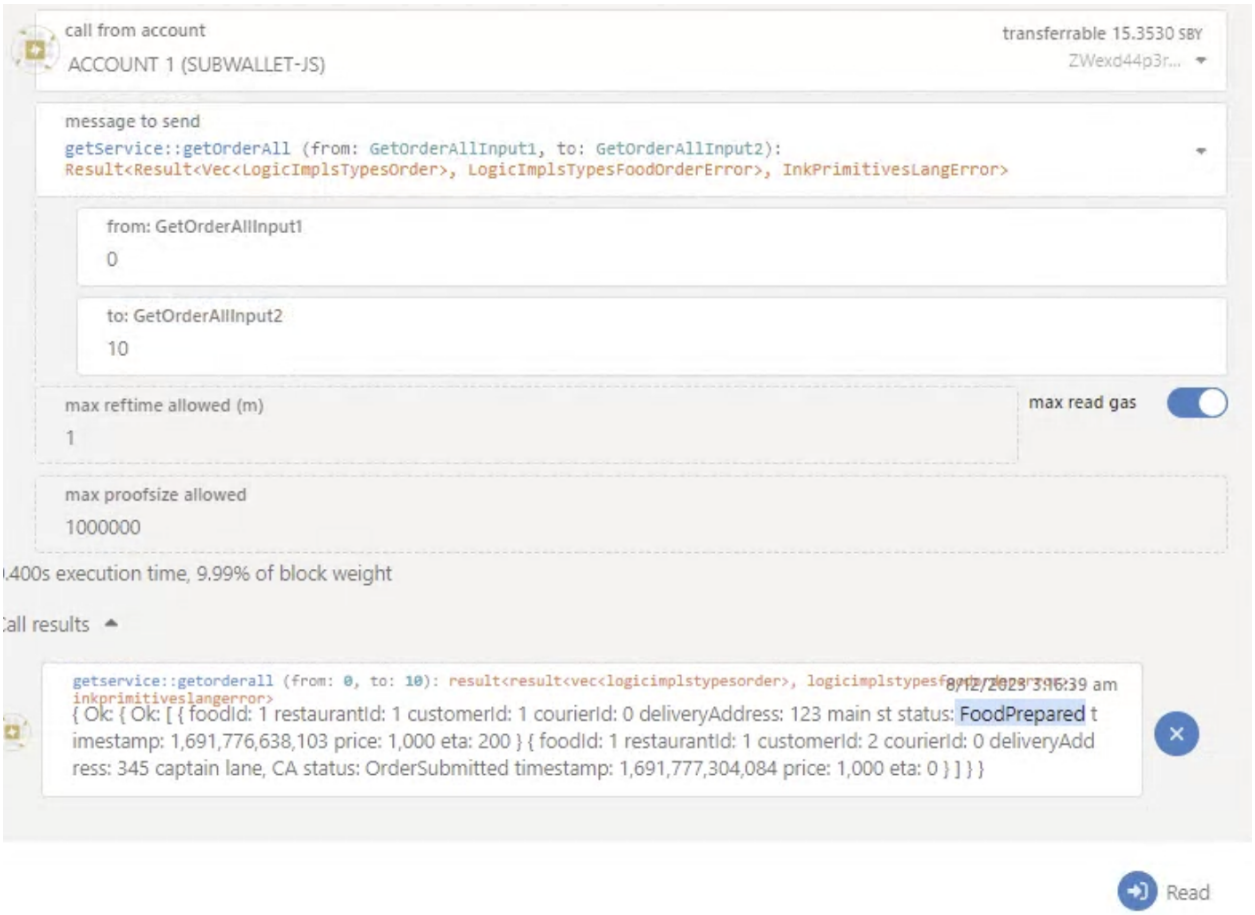


7. Finish cooking

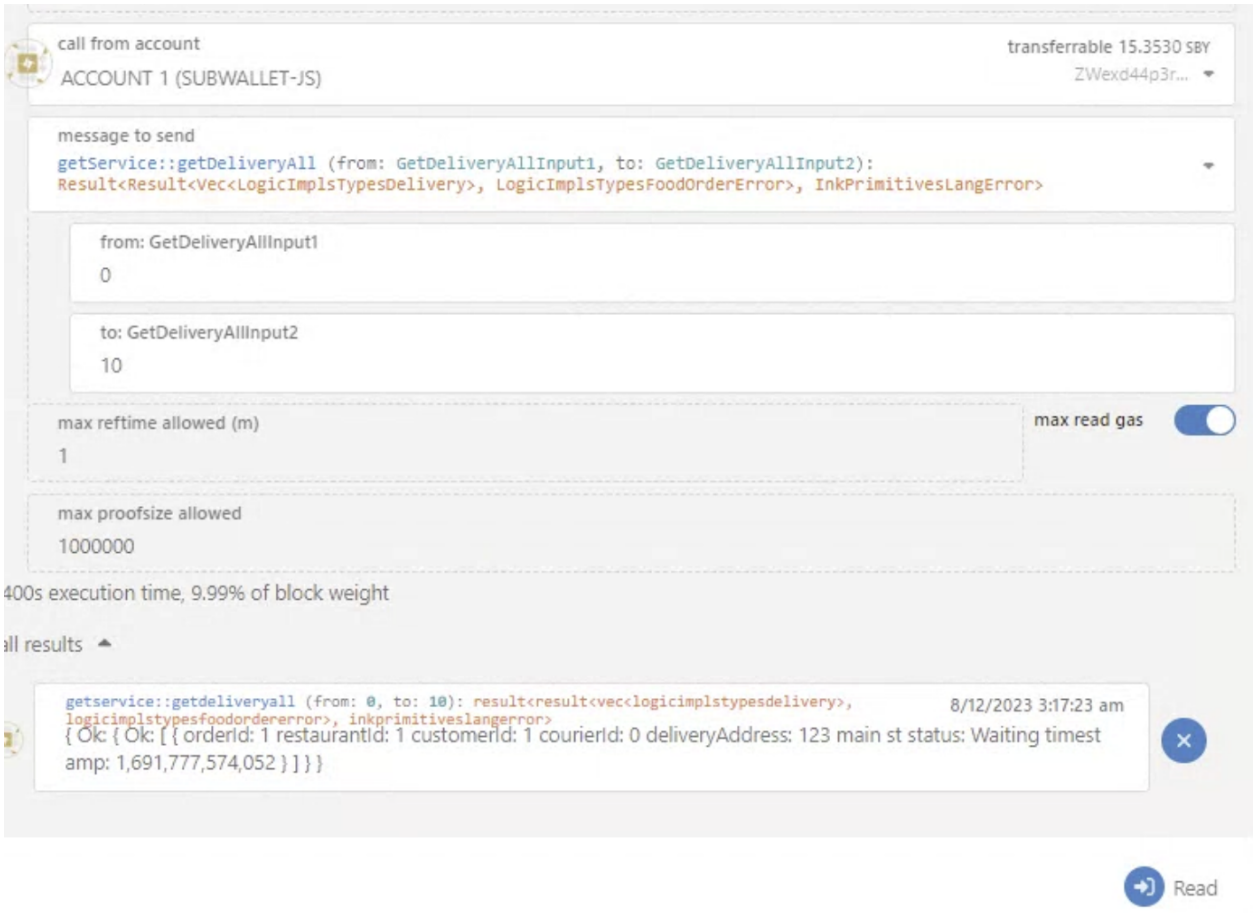
After food is cooked, you can call finish_cook.



And check the order status.

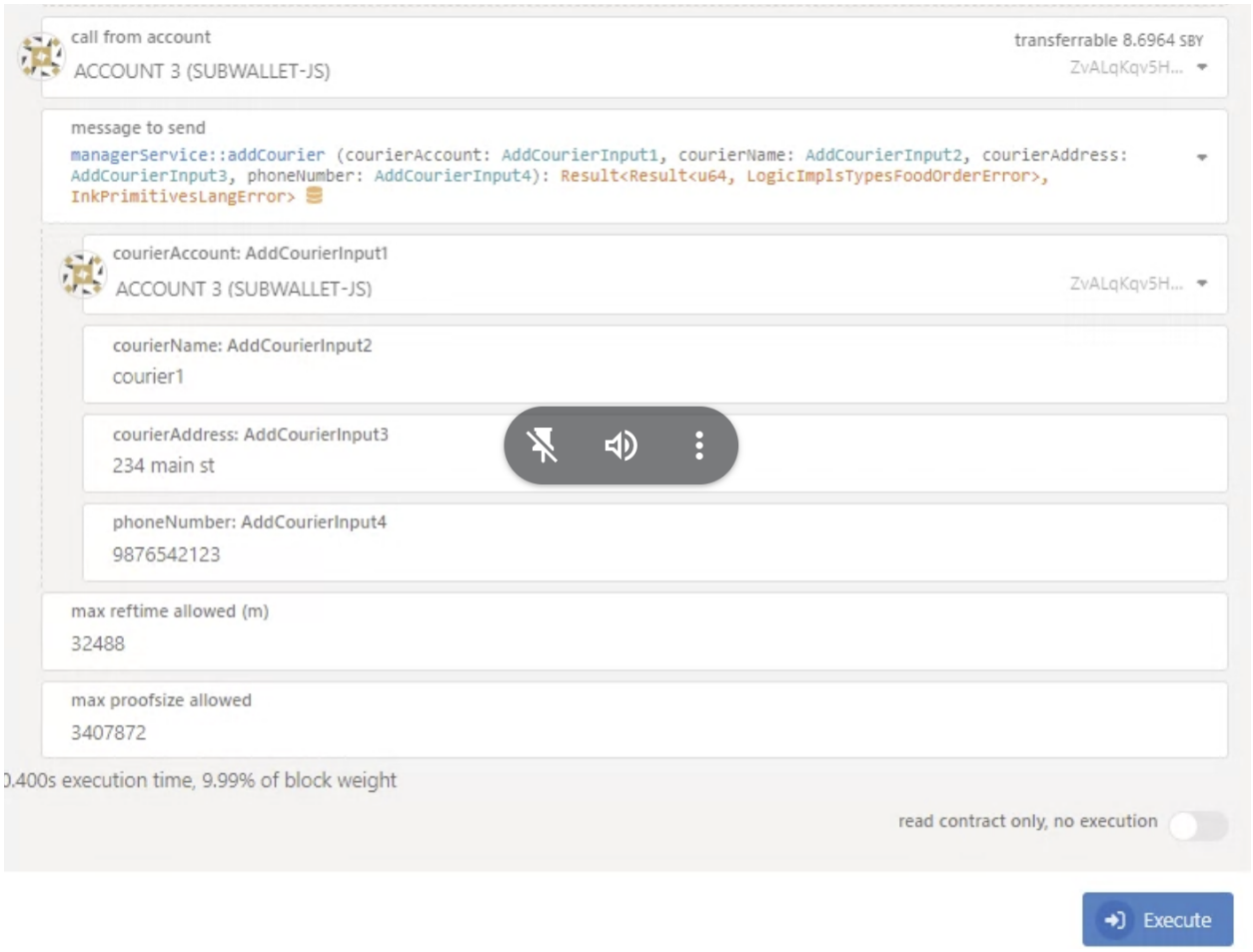


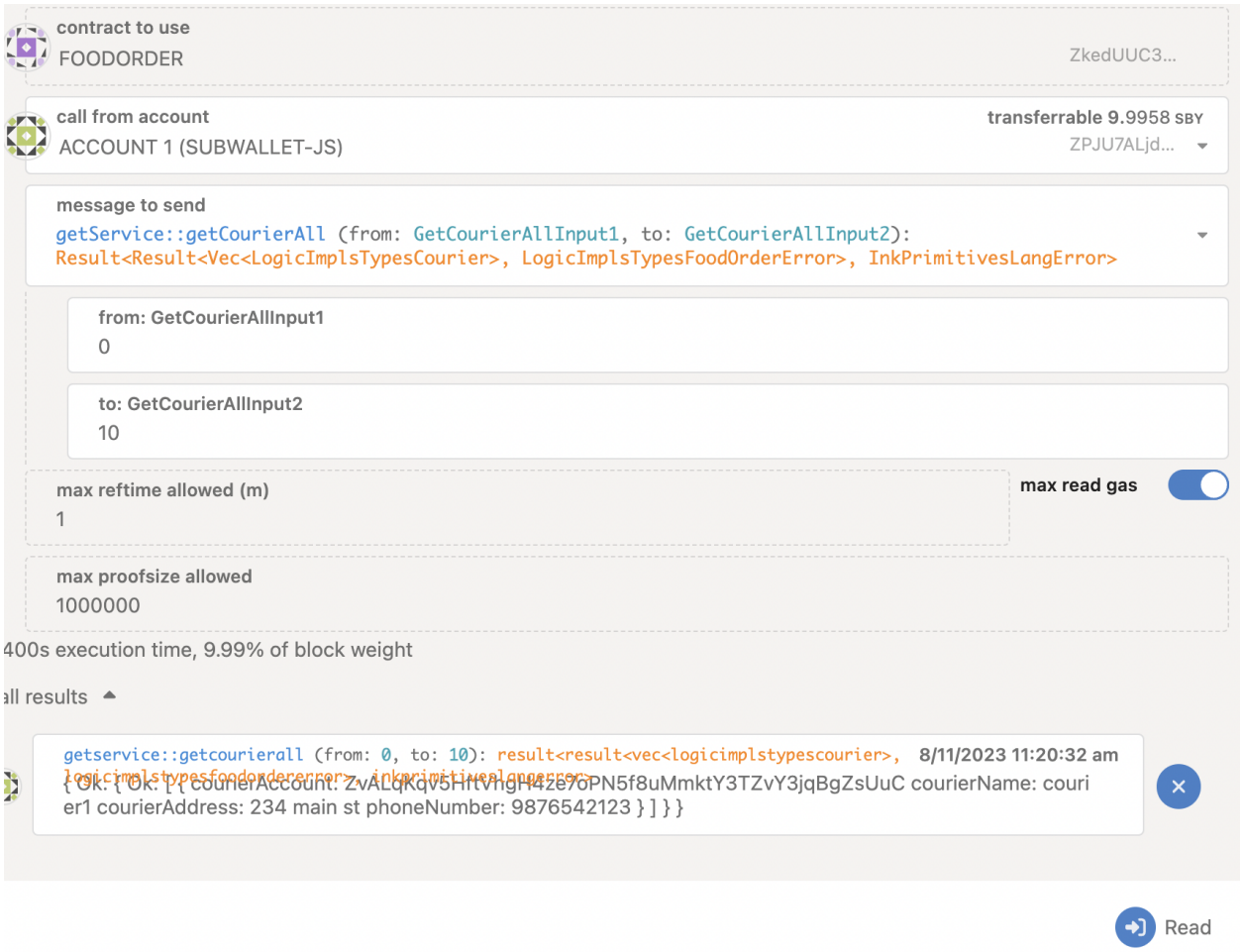
Once the restaurant confirms an order, it generally calls a courier to let him deliver the food. Let’s check the delivery status too.



8. Add a courier

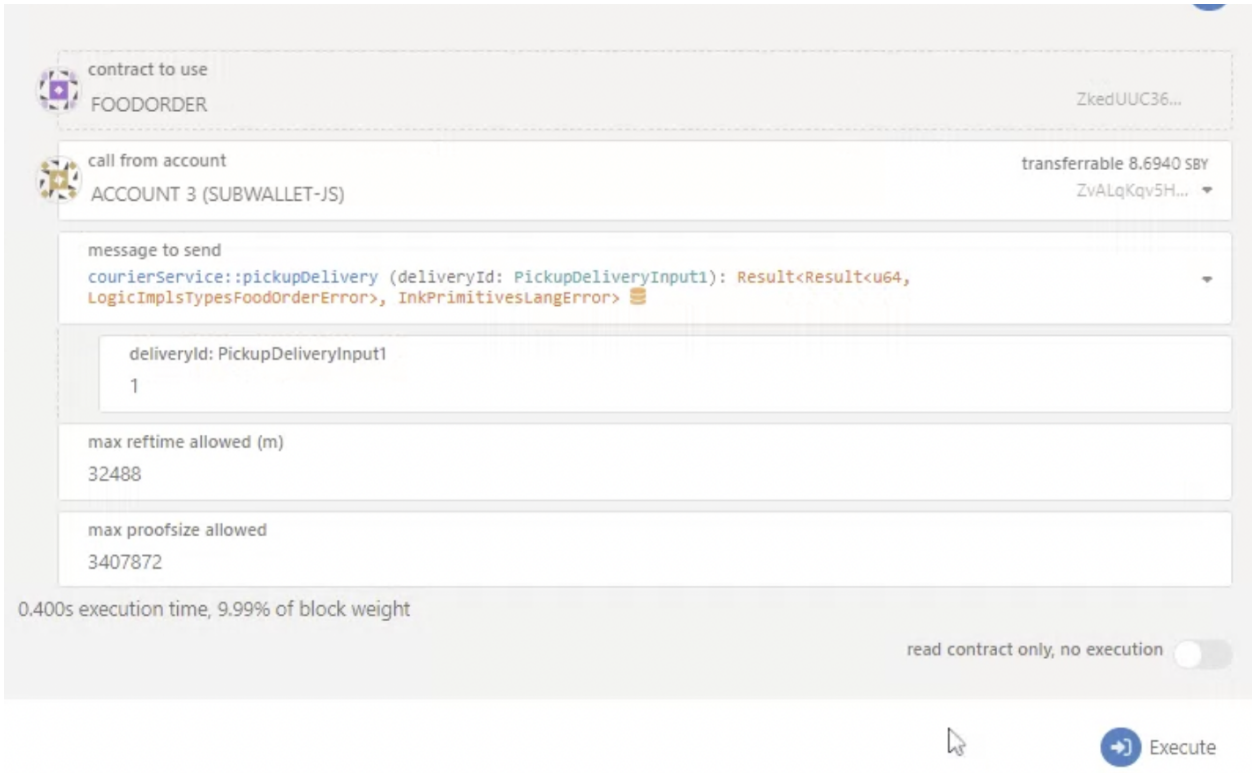
The courier can be registered by calling `add_courier` message.

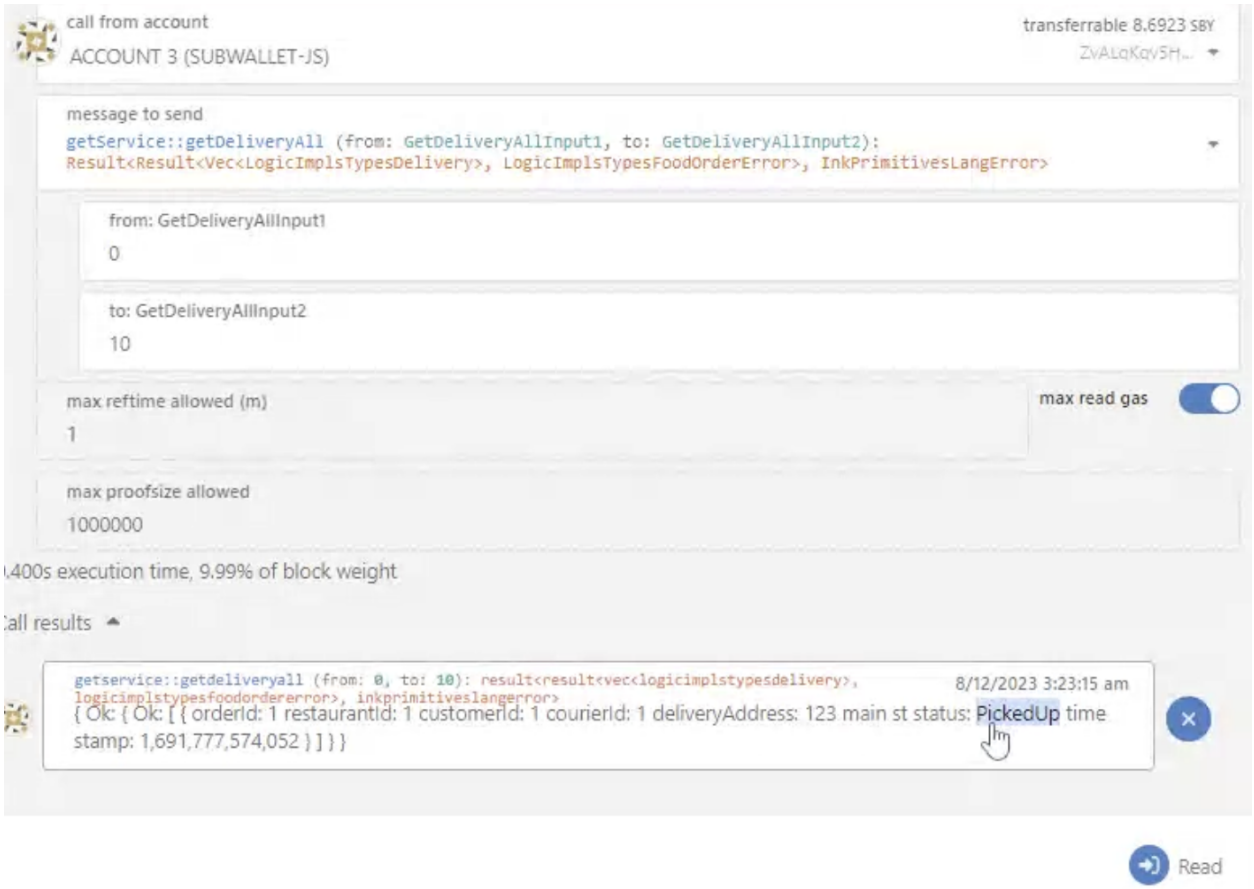

call from account
ACCOUNT 3 (SUBWALLET-JS)
transferrable 8.6964 SBY
ZvALqKqv5H...
message to send
managerService::addCourier (courierAccount: AddCourierInput1, courierName: AddCourierInput2, courierAddress: AddCourierInput3, phoneNumber: AddCourierInput4): Result<Result<u64, LogicImplsTypesFoodOrderError>, InkPrimitivesLangError>
courierAccount: AddCourierInput1
ACCOUNT 3 (SUBWALLET-JS)
ZvALqKqv5H...
courierName: AddCourierInput2
courier1
courierAddress: AddCourierInput3
234 main st
phoneNumber: AddCourierInput4
9876542123
max reftime allowed (m)
32488
max proofsize allowed
3407872
.400s execution time, 9.99% of block weight
read contract only, no execution
Execute



9. Pickup a delivery

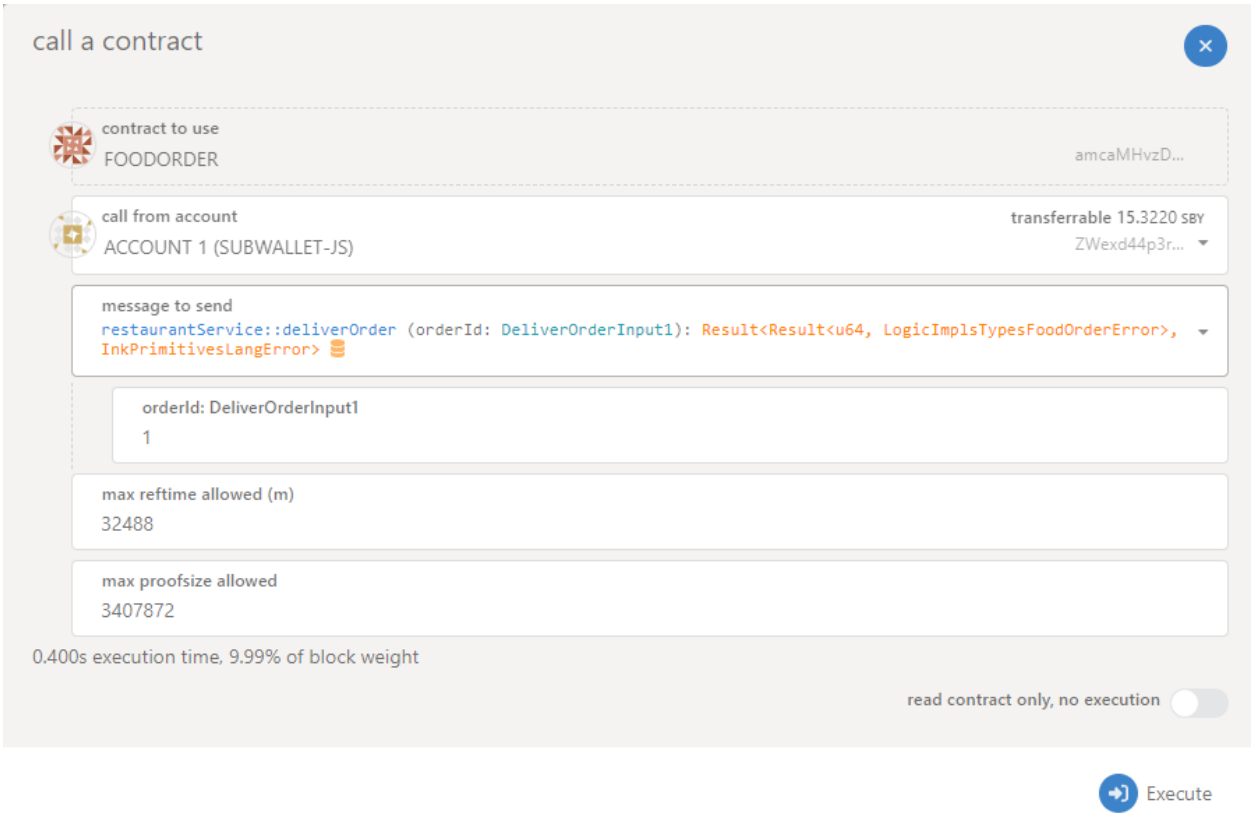
Once a courier arrives at the restaurant, then he picks up a delivery.
This is implemented in the `pickupDelivery` message.

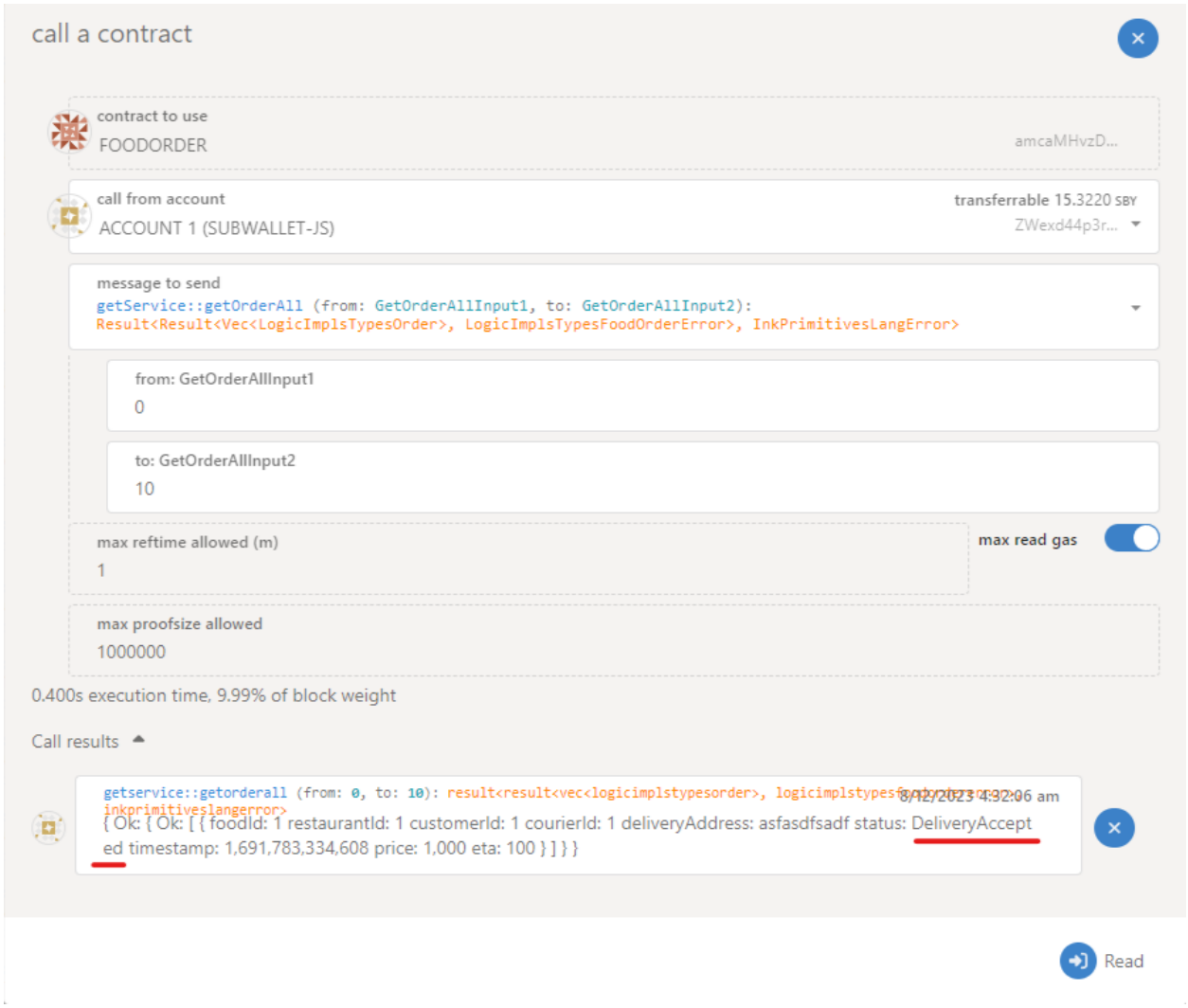

contract to use
FOODORDER
ZkedUUC36...
call from account
ACCOUNT 3 (SUBWALLET-JS)
transferrable 8.6940 SBY
ZvALqKqv5H...
message to send
courierService::pickupDelivery (deliveryId: PickupDeliveryInput1): Result<Result<u64, LogicImplsTypesFoodOrderError>, InkPrimitivesLangError>
deliveryId: PickupDeliveryInput1
1
max reftime allowed (m)
32488
max proofsize allowed
3407872
0.400s execution time, 9.99% of block weight
read contract only, no execution
Execute


call from account
ACCOUNT 3 (SUBWALLET-JS)
transferrable 8.6923 SBY
ZvALqKqv5H...
message to send
getService::getDeliveryAll (from: GetDeliveryAllInput1, to: GetDeliveryAllInput2): Result<Result<Vec<LogicImplsTypesDelivery>, LogicImplsTypesFoodOrderError>, InkPrimitivesLangError>
from: GetDeliveryAllInput1
0
to: GetDeliveryAllInput2
10
max reftime allowed (m)
1
max read gas
max proofsize allowed
1000000
.400s execution time, 9.99% of block weight
all results
getservice::getdeliveryall (from: 0, to: 10): result<result<vec<logicimplstypesdelivery>, logicimplstypesfoodordererror>, inkprimitiveslangerror>
8/12/2023 3:23:15 am
{ Ok: { Ok: [{ orderId: 1 restaurantId: 1 customerId: 1 courierId: 1 deliveryAddress: 123 main st status: PickedUp time stamp: 1,691,777,574,052 }] } }
Read

10. Deliver an order

It means that the restaurant finally gives the food to a courier and the courier starts to carry it to its destination.





11. Accept a delivery.

Once the delivery is carried to the customer, then the customer will check the order and accept it.

contract to use
FOODORDER
ZkedUUC3...

call from account
ACCOUNT 1 (SUBWALLET-JS)
transferrable 9.9958 SBY
ZPJU7ALjd...

message to send
customerService::acceptDelivery (deliveryId: AcceptDeliveryInput1): Result<Result<u64, LogicImplsTypesFoodOrderError>, InkPrimitivesLangError>

deliveryId: AcceptDeliveryInput1
1

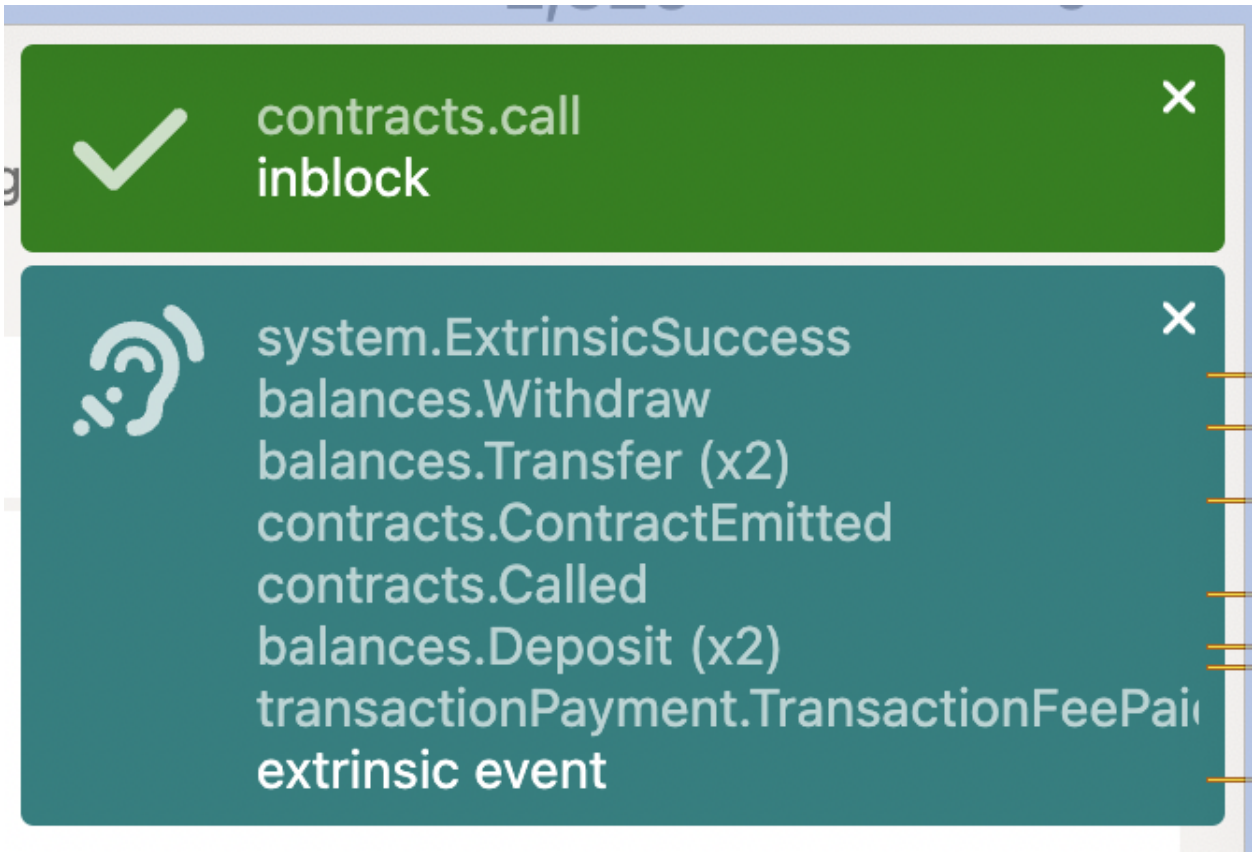
max reftime allowed (m)
32488

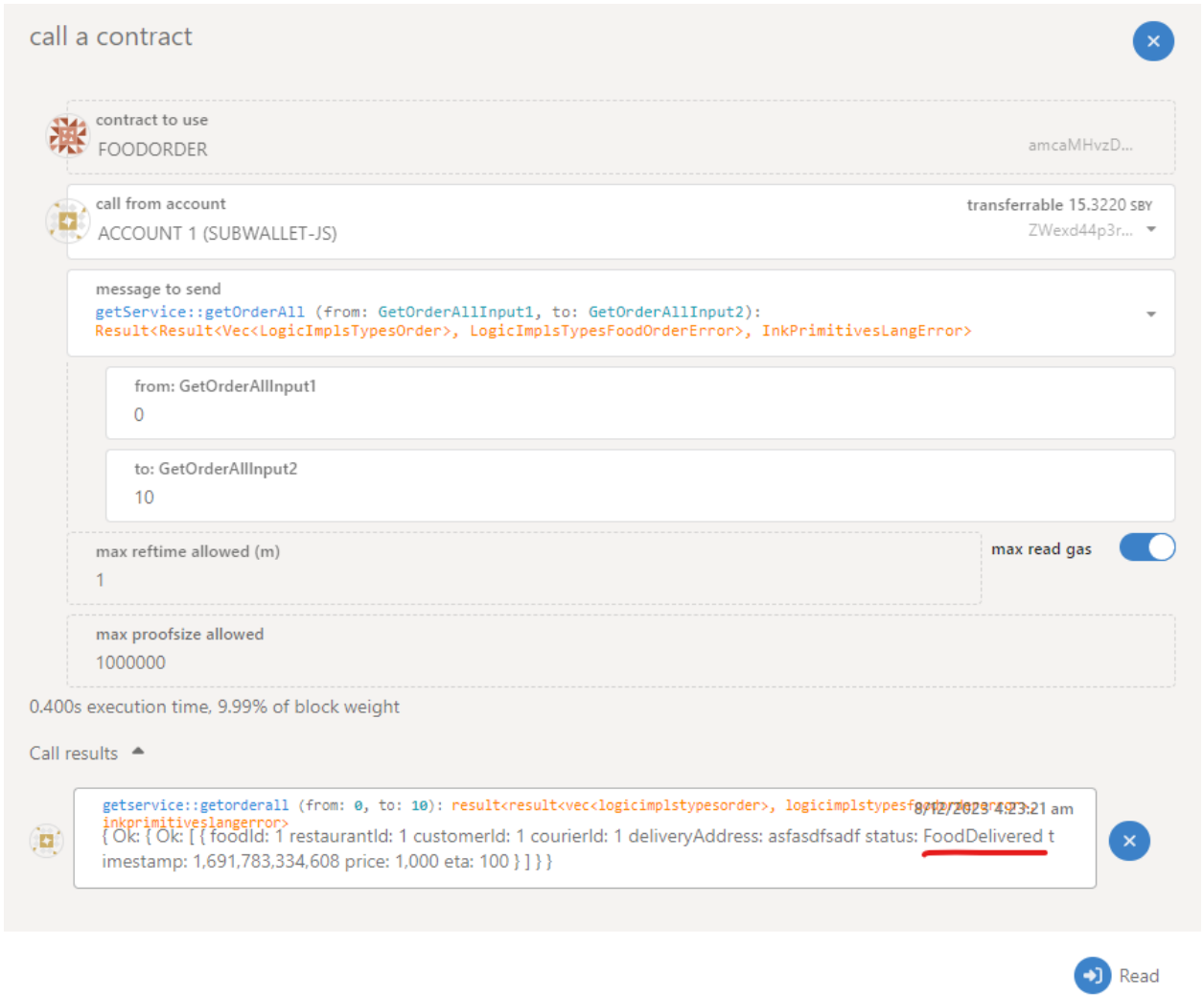
max proofsize allowed
3407872

0.400s execution time, 9.99% of block weight

read contract only, no execution

Execute




call a contract
contract to use
FOODORDER
amcaMHvzD...
call from account
ACCOUNT 1 (SUBWALLET-JS)
transferrable 15.3220 SBY
ZWexd44p3r...
message to send
getService::getOrderAll (from: GetOrderAllInput1, to: GetOrderAllInput2):
Result<Result<Vec<LogicImplsTypesOrder>, LogicImplsTypesFoodOrderError>, InkPrimitivesLangError>
from: GetOrderAllInput1
0
to: GetOrderAllInput2
10
max reftime allowed (m)
1
max read gas
max proofsize allowed
1000000
0.400s execution time, 9.99% of block weight
Call results
getservice::getorderall (from: 0, to: 10): result<result<vec<logicimplstypesorder>, logicimplstypesfoodorderer
inkprimitiveslangerror>
8/12/2023 4:23:21 am
{ Ok: { Ok: [{ foodId: 1 restaurantId: 1 customerId: 1 courierId: 1 deliveryAddress: asfasdfsadf status: FoodDelivered t
imestamp: 1,691,783,334,608 price: 1,000 eta: 100 }] } }
Read