

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.

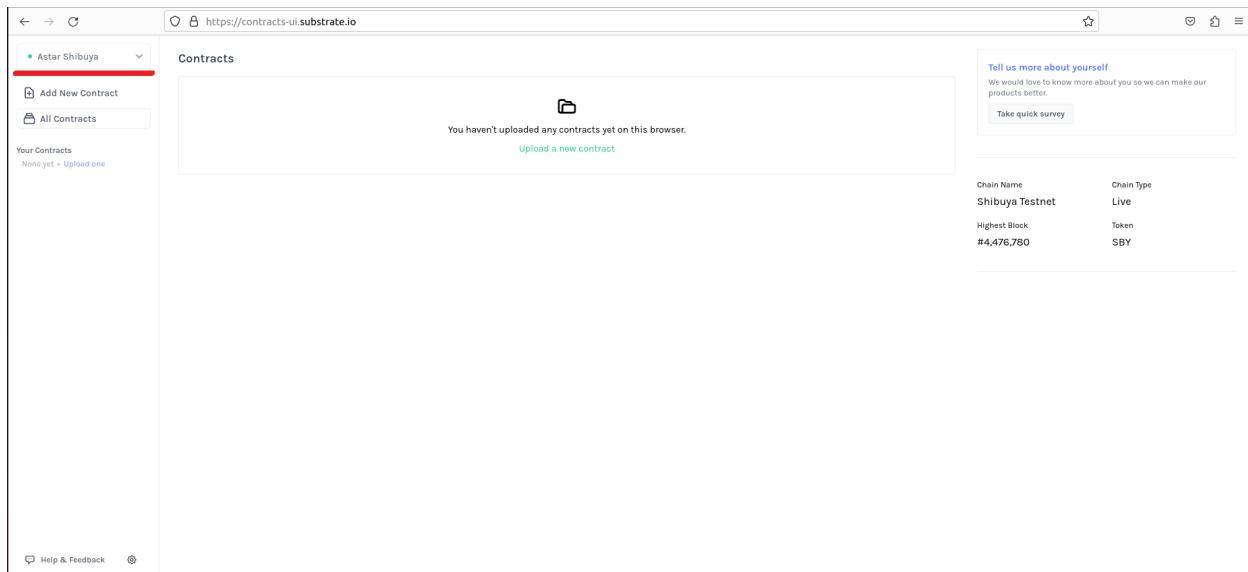
[Contracts-UI](#) is a web application for deploying Wasm smart contracts on Substrate chains. [Polkadot.js](#) is an effort to provide a collection of tools, utilities and libraries for interacting with the Polkadot network from JavaScript.

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.

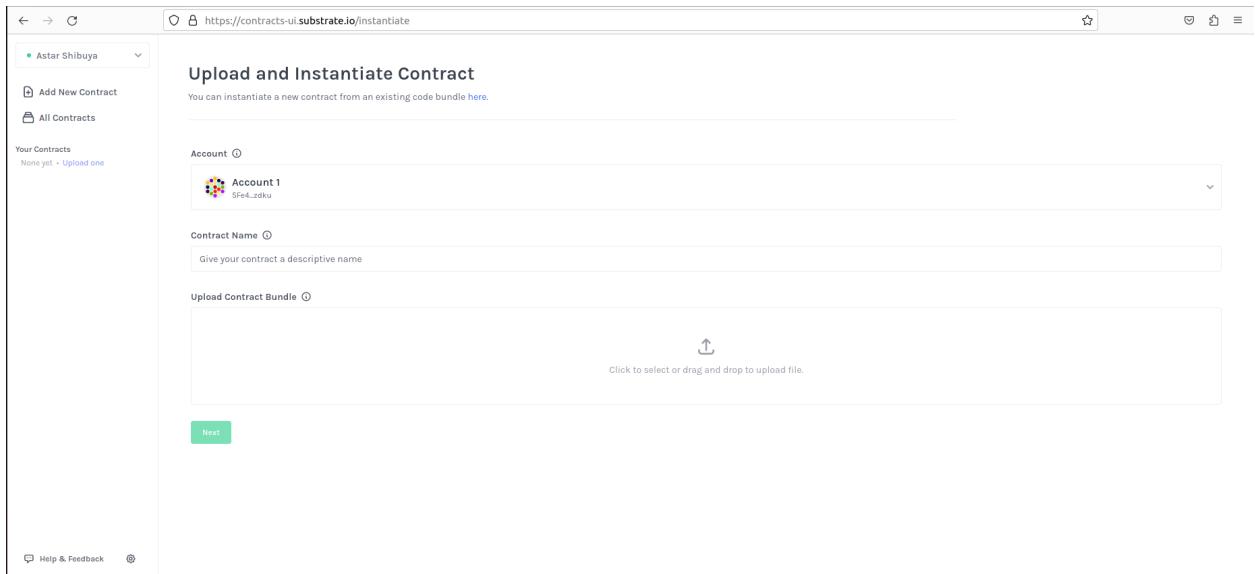
Working with Contracts-UI

Upload and instantiate

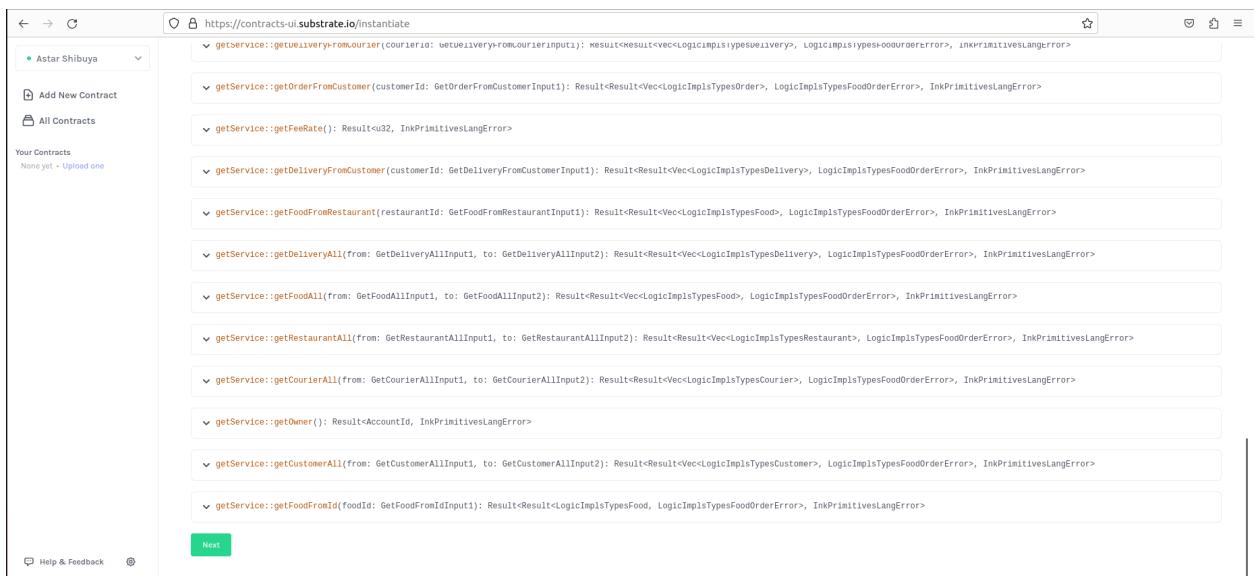
1. Change the network to Astar Shibuya.



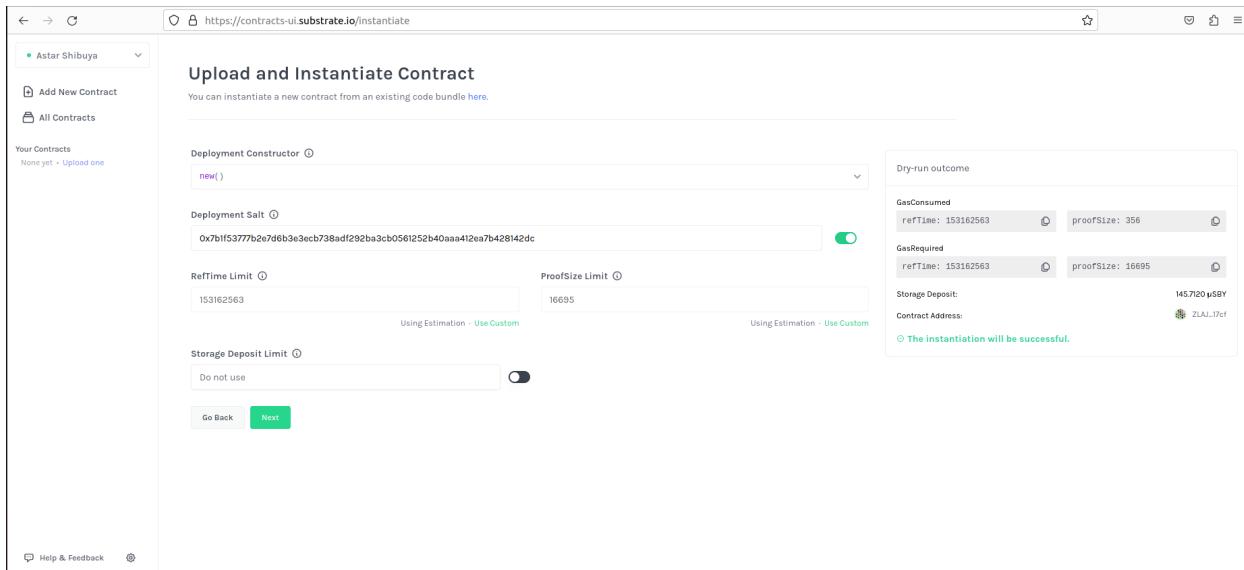
2. Click the “Upload a new contract” button.



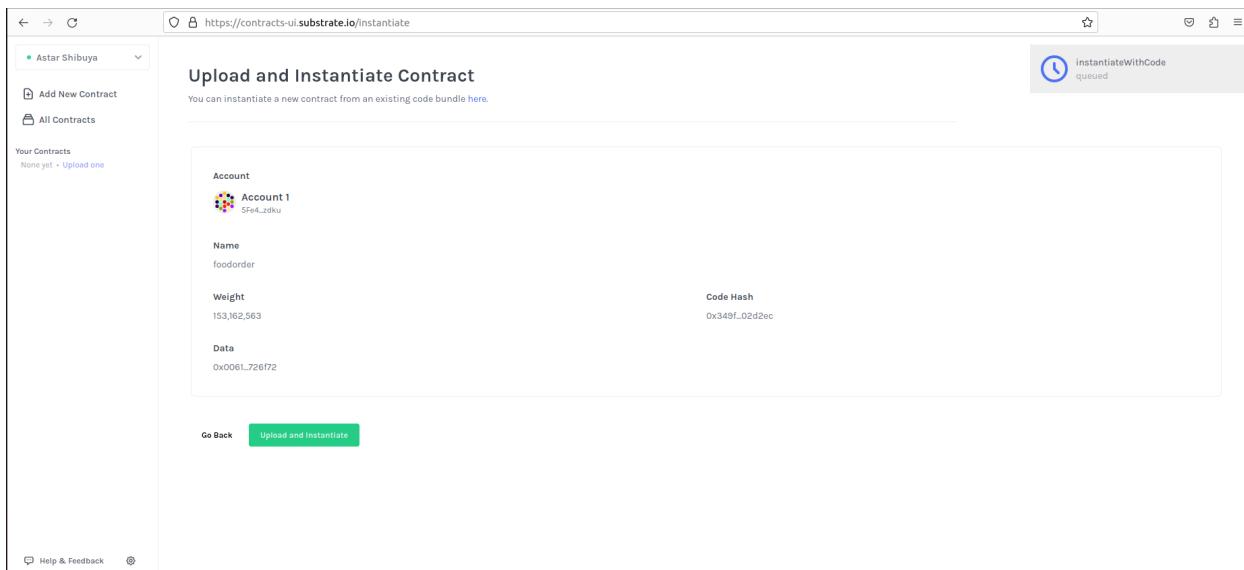
3. Drag and drop the “foodorder.contract” file and click the “Next” button.



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](#).

The screenshot shows the Shibuya web interface. On the left, there's a sidebar with navigation links: Assets (which is selected and highlighted in blue), Dashboard, dApp Staking, NFT, Ecosystem, and Forum. The main content area is titled "Assets" and "Astar Native Account". It shows an account named "Account 1 (subwallet-js)" with the address "ZPJU7A.....Yrakso" and a balance of "0 USD". Below this, it shows an asset entry for "SBY" with the amount "10 SBY" and the note "(Transferable) 10 SBY".

2. Restaurant registers itself into the smart contract storage.

The screenshot shows the contracts-ui substrate interface. On the left, there's a sidebar with "Astar Shibuya" selected, "Add New Contract", "All Contracts", and "Foodorder". The main content area shows a form for registering a new restaurant. The "Message to Send" field contains the call: "managerService::addRestaurant(restaurantAccount: AddRestaurantInput1, restaurantName: AddRestaurantInput2, restaurantAddress: AddRestaurantInput3, phoneNumber: AddRestaurantInput4)". The "Dry-run outcome" panel shows the result: "Contract call will be successful!". The "Execution result" panel shows the JSON response: "{'Ok': '1'}". The "GasConsumed" panel shows "refTime: 3668784583" and "proofSize: 82950". The "GasRequired" panel shows "refTime: 4793859072" and "proofSize: 125952". The "StorageDeposit" panel shows "charge: 76.0000 nSBY". The "Transactions log" panel shows "No transactions yet." At the bottom, there are buttons for "Call contract" and "Help & Feedback".

The screenshot shows the contracts-ui substrate interface. On the left, there's a sidebar with 'Astar Shibuya' selected, 'Add New Contract', 'All Contracts', and 'Your Contracts' (Foodorder). The main area has a form for sending a message to the 'Restaurant' contract. The message payload is:

```
managerService::addRestaurant(restaurantAccount: AddRestaurantInput1, restaurantName: AddRestaurantInput2, restaurant: AddRestaurantInput3, restaurantAddress: AddRestaurantInput4, phoneNumber: AddRestaurantInput5)
```

The inputs are filled with values: restaurantAccount, restaurantName, restaurantAddress, and phoneNumber. Below the inputs are sections for 'ReffTime Limit' (4793859072), 'ProofSize Limit' (125952), and 'Storage Deposit Limit' (set to 'Do not use').

To the right, a 'Dry-run outcome' panel shows the execution result of the call to 'addRestaurant'. It includes fields like 'GasConsumed', 'GasRequired', 'StorageDeposit', and a 'Transactions log' section indicating 'No transactions yet'.

A modal window titled 'Signature request' is open, showing the same message payload and a button to 'Approve' it. The modal also displays the account 'Restaurant (YZuT...ZSDE)' and the URL 'contracts-ui.substrate.io'.

This screenshot shows the result of the signature request from the previous step. The干run outcome now indicates a successful call ('call success') to the 'addRestaurant' function. The execution result shows the transaction details: 'RestaurantAlreadyExist' (gas consumed: 3102290561), 'balances:Withdraw' (gas required: 4793859072), and 'transactionPayment:TransactionFeePaid' (gas required: 125952).

The 'Transactions log' shows the event 'ManagerService::add_restaurant()' occurring at 22/8/2023, 12:32:55 pm. The 'CONTRACT EVENTS' section lists 'AddRestaurantEvent' with values: restaurantId: '1', restaurantName: 'RestaurantA', restaurantAddress: '123 Main St', phoneNumber: '123456789'. The 'GENERIC EVENTS' section shows 'balances:Withdraw'.

3. The Courier registers itself into the smart contract.

The screenshot shows the contracts-ui substrate interface for the Astar Shibuya network. The user is interacting with a smart contract to add a new courier. The 'Dry-run outcome' section indicates a successful call with the message: 'Contract call will be successful!'. The 'Execution result' shows a JSON object with 'Ok: '1''. The 'GasConsumed' section shows a refTime of 3674238680 and a proofSize of 82950. The 'GasRequired' section shows a refTime of 4793859072 and a proofSize of 125952. The 'StorageDeposit' section shows a charge of 77.0000 nSBY. The 'Transactions log' shows a single entry from 22/8/2023 at 12:32:55 pm: 'ManagerService::add_restaurant()'. Below the log, a 'CONTRACT EVENTS' section shows an 'AddRestaurantEvent' with details: restaurantId "T", restaurantName "RestaurantA", restaurantAddress "123 Main St", phoneNumber "123456789". At the bottom, there is a green 'Call contract' button.

This screenshot shows the same interface after the user has clicked the 'Call contract' button. A modal dialog titled 'Signature request' has appeared in the center. The dialog is titled 'Signature request' and includes the text 'You are approving a request with the following account'. It shows the account 'Courier (WKPT_3NEY)' with a green checkmark icon. At the bottom of the dialog are two buttons: 'Cancel' and 'Approve', with 'Approve' being highlighted in blue. The background of the main interface shows the same dry-run outcome and transaction log as the previous screenshot.

4. A Customer registers itself into the smart contract.

The screenshot shows the contracts-ui substrate interface for the Astar Shibuya network. A modal window titled "Signature request" is open, indicating that a request is being approved. The main interface displays a successful dry-run outcome for the "customerService::addCustomer" call. The execution result is shown as a JSON object:

```
{
  "Ok": "1"
}
```

Key metrics shown include:

- GasConsumed**: refTime: 3491983360, proofSize: 89149
- GasRequired**: refTime: 4793859672, proofSize: 125952
- StorageDeposit**: charge: 79.0000 nSBY

The transaction log shows the event: ManagerService::add_courier(). The contract events section lists the AddCourierEvent with the following details:

- courierId: "1"
- courierName: "CourierA"
- courierAddress: "234 Downtown St"
- phoneNumber: "+654987321"

The screenshot shows the contracts-ui substrate interface for the Astar Shibuya network. A modal window titled "Signature request" is open, showing the approval process for a request from the "Customer" account. The main interface displays a successful dry-run outcome for the "customerService::addCustomer" call. The execution result is shown as a JSON object:

```
{
  "Ok": "1"
}
```

Key metrics shown include:

- GasConsumed**: refTime: 3491983360
- GasRequired**: refTime: 4793859672
- StorageDeposit**: charge: 79.0000 nSBY

The transaction log shows the event: ManagerService::add_courier(). The contract events section lists the AddCourierEvent with the following details:

- courierId: "1"
- courierName: "CourierA"
- courierAddress: "234 Downtown St"
- phoneNumber: "+654987321"

5. The Restaurant adds one food.

The screenshot shows the contracts-ui substrate interface for the Foodorder contract. On the left, the sidebar shows 'Astar Shibuya' selected, with options to 'Add New Contract' and 'All Contracts'. Under 'Your Contracts', 'Foodorder' is listed. The main area has a 'Caller' dropdown set to 'Restaurant' (56n_k_yfJK). A 'Message to Send' section contains the message: `restaurantService::addFood(foodName: AddFoodInput1, description: AddFoodInput2, price: AddFoodInput3, eta: AddFoodInput4)`. Below it are input fields for `FoodName`, `description`, `price`, and `eta`. Under 'ReffTime Limit' and 'ProofSize Limit', there are input fields with values 4793859072 and 125952 respectively. A 'Storage Deposit Limit' toggle switch is turned off. At the bottom is a green 'Call contract' button. To the right, the 'Dry-run outcome' panel shows a successful execution result with an OK status, gas consumption details, and a storage deposit charge of 63.0000 nSBY. The 'Transactions log' panel shows the timestamp 22/8/2023, 12:38:55 pm and the event `CustomerService::add_customer()`. The bottom right shows the timestamp 22/8/2023, 12:34:56 nm.

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

This screenshot shows the same interface after calling the `getFoodFromId` message. The 'Caller' is now set to 'Customer' (56o2_p5gP). The 'Message to Send' section contains the message: `getService::getFoodFromId(foodId: GetFoodFromIdInput1): Result<Result<LogicImplsTypesFood, LogicImplsTypesFoodOrderError>`. The `foodId` field is filled with the value '1'. The 'Outcome' panel on the right shows the return value: `{ Ok: { foodName: 'FoodA', restaurantId: '1', description: 'FoodA', price: '50', eta: '100', timestamp: '1,692,679,230,628' }, }`. The 'Transactions log' panel shows the timestamp 22/8/2023, 12:42:43 pm and the event `CustomerService::submit_order()`. The bottom right shows the timestamp 22/8/2023, 12:42:43 pm.

7. The restaurant adds one more food.

Dry-run outcome

```
Contract call will be:
{ "ok": "2" }
```

Execution result

```
{ "ok": "2" }
```

GasConsumed

```
refTime: 3718528229
```

GasRequired

```
refTime: 4793859672
```

StorageDeposit

```
charge: 61.0000 nSBY
```

Transactions log

```
22/8/2023, 12:42:43 pm CustomerService::submit
```

CONTRACT EVENTS

```
SubmitOrderEvent
    foodId "1"
    restaurantId "1"
    customerId "1"
```

Signature request

You are approving a request with the following account

Restaurant (YZuT...ZSDE)

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

Outcome

Return value

```
{
  "ok": [
    {
      "foodName": "FoodA",
      "restaurantId": "1",
      "description": "FoodA",
      "price": "50",
      "eta": "100",
      "timestamp": "1,692,679,230,620",
    },
    {
      "foodName": "FoodB",
      "restaurantId": "1",
      "description": "FoodB",
      "price": "100",
      "eta": "200",
      "timestamp": "1,692,679,470,628",
    }
  ]
}
```

Transactions log

```
22/8/2023, 12:44:44 pm RestaurantService::add_food()
```

CONTRACT EVENTS

```
AddFoodEvent
    foodId "2"
    foodName "FoodB"
    restaurantId "1"
```

9. The customer submits an order.

The screenshot shows the Substrate Contracts UI interface. On the left, there's a sidebar with 'Astar Shibuya' selected, 'Add New Contract', 'All Contracts', and a list of 'Your Contracts' including 'Foodorder'. The main area has a 'Caller' dropdown set to 'Customer' (5eo2-psgP). A 'Message to Send' section contains the message: `customerService::submitOrder(foodId: SubmitOrderInput1, deliveryAddress: SubmitOrderInput2): Result<Result<u64, Logic`. Below it, 'foodId: SubmitOrderInput1' is set to '4793859072' and 'deliveryAddress: SubmitOrderInput2' is set to '324 Main St'. There are sections for 'ReffTime Limit' (4793859072), 'ProofSize Limit' (125952), 'Storage Deposit Limit' (Value: 0.0000000000000005 SBY, using Estimation), and a 'Call contract' button. To the right, the 'Dry-run outcome' panel shows a green success message: 'Contract call will be successful!', with execution results including gas consumption, storage deposit, and transaction logs.

10. Check the order status.

This screenshot shows the same Substrate Contracts UI interface. The 'Message to Send' section now contains the message: `getService::getOrderFromId(orderId: GetOrderFromIdInput1): Result<Result<LogicImplTypesOrder, LogicImplTypesFoodOrd`. Below it, 'orderId: GetOrderFromIdInput1' is set to '1'. The 'Outcome' panel on the right shows the return value: `{ Ok: { foodId: '1', restaurantId: '1', customerId: '1', courierId: '9', deliveryAddress: '324 Main St', status: 'OrderSubmitted', timestamp: '1,692,679,398,328', price: '50', eta: '9' }, }`. The transaction log shows a 'RestaurantService::add_food()' event.

The screenshot shows the Astar Shibuya substrate UI interface. On the left, there's a sidebar with options like 'Add New Contract', 'All Contracts', and 'Your Contracts' (Foodorder). The main area has a 'Caller' section showing 'Customer' (5eo2...psgP). Below it is a 'Message to Send' section with the following code:

```
getService::getOrderAll(from: GetOrderAllInput1, to: GetOrderAllInput2): Result<Result<Vec<logicImplsTypeOrder>, Log> {
    from: GetOrderAllInput1
    1
    to: GetOrderAllInput2
    10
}
```

To the right, there are several panels: 'Outcome' showing a JSON response with an 'Ok' key containing an array of order objects; 'Return value' showing the same JSON; 'Transactions log' showing a single event 'RestaurantService::add_food()' at 22/8/2023, 12:44:44 pm; and a 'CONTRACT EVENTS' panel for 'AddFoodEvent'.

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

This screenshot shows the Astar Shibuya substrate UI. The sidebar includes 'Add New Contract', 'All Contracts', and 'Your Contracts' (Foodorder). The 'Caller' section shows 'Restaurant' (6HK...yJk). The 'Message to Send' section contains the following code:

```
restaurantService::confirmOrder(orderId: ConfirmOrderInput1, eta: ConfirmOrderInput2): Result<Result<u64, logicImplsT>, Log> {
    orderId: ConfirmOrderInput1
    1
    eta: ConfirmOrderInput2
    120
}
```

Below the message input are fields for 'RefTime Limit' (4793859072) and 'ProofSize Limit' (125552). At the bottom is a 'Call contract' button. To the right, there are panels for 'Dry-run outcome' (Contract call will be successful), 'Execution result' (Ok: '1'), 'GasConsumed' (refTime: 4376092637, proofSize: 89267), 'GasRequired' (refTime: 4793859072, proofSize: 125952), 'StorageDeposit' (charge: 74.0000 nSBY), and 'Transactions log' showing the 'RestaurantService::add_food()' event.

12. Check the order status and delivery status.

The screenshot shows a web-based interface for interacting with a blockchain contract. On the left, a sidebar lists "Astar Shibuya" and "Foodorder". The main area has tabs for "Caller" (set to "Restaurant") and "Outcome".

Caller: Restaurant (SEhK_yfJK)

Message to Send:

```
getService::getOrderFromId(orderId: GetOrderFromIdInput1): Result<Result<LogicImplsTypesOrder, LogicImplsTypesFoodOrd>, Error>
```

orderId: GetOrderFromIdInput1

1

Outcome:

```
{
  Ok: {
    orderId: '1',
    restaurantId: '1',
    customerId: '1',
    courierId: '0',
    deliveryAddress: '324 Main St',
    status: 'OrderConfirmed',
    timestamp: '1,692,679,550,325',
    price: '50',
    eta: '120',
  }
}
```

Transactions log:

22/8/2023, 12:46:56 pm
RestaurantService::confirm_order()

CONTRACT EVENTS:

- ConfirmOrderEvent
 - orderId: "1"
 - eta: "120"
- RequestDeliveryEvent
 - orderId: "1"
 - restaurantId: "1"
 - customerId: "1"
 - deliveryAddress: "324 Main St", status: "OrderConfirmed", timestamp: "1,692,679,550,325"

The screenshot shows a similar interaction between the Restaurant and Foodorder contracts.

Caller: Restaurant (SEhK_yfJK)

Message to Send:

```
getService::getDeliveryAll(from: GetDeliveryAllInput1, to: GetDeliveryAllInput2): Result<Result<Vec<LogicImplsTypesOrder>, Error>
```

from: GetDeliveryAllInput1

1

to: GetDeliveryAllInput2

10

Outcome:

```
{
  Ok: [
    {
      orderId: '1',
      restaurantId: '1',
      customerId: '1',
      courierId: '0',
      deliveryAddress: '324 Main St',
      status: 'OrderConfirmed',
      timestamp: '1,692,679,602,143',
    },
  ],
}
```

Transactions log:

22/8/2023, 12:46:56 pm
RestaurantService::confirm_order()

CONTRACT EVENTS:

- ConfirmOrderEvent
 - orderId: "1"
 - eta: "120"
- RequestDeliveryEvent
 - orderId: "1"
 - restaurantId: "1"
 - customerId: "1"
 - deliveryAddress: "324 Main St", status: "OrderConfirmed", timestamp: "1,692,679,602,143"

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

The screenshot shows the Substrate Contracts UI interface. On the left, there's a sidebar with 'Astar Shibuya' selected, 'Add New Contract', 'All Contracts', 'Your Contracts' (Foodorder), and a 'Help & Feedback' link. The main area has a 'Caller' section showing 'Restaurant' (SEHK-yfJK). Below it is a 'Message to Send' section with the code: `restaurantService::finishCook(orderId: FinishCookInput1): Result<Result<u64, LogicImplsTypesFoodOrderError>, InkPrim1`. The 'orderId' field contains '1'. To the right, there are sections for 'ReffTime Limit' (4793859072) and 'ProofSize Limit' (125952). Underneath these are 'Using Estimation - Use Custom' buttons. A 'Storage Deposit Limit' section with a 'Do not use' toggle is also present. A large green 'Call contract' button is at the bottom. On the right side, there's a 'Dry-run outcome' panel with a success message: 'Contract call will be successful!', an 'Execution result' panel showing a JSON object with 'Ok: 1', and other panels for 'GasConsumed', 'GasRequired', and 'StorageDeposit'. Below these is a 'Transactions log' panel showing a single event: '22/8/2023, 12:46:56 pm RestaurantService::confirm_order()'.

14. Check the order status.

This screenshot shows the same Substrate Contracts UI interface. The 'Message to Send' section now contains the code: `getService::getOrderFromId(orderId: GetOrderFromIdInput1): Result<Result<LogicImplsTypesOrder, LogicImplsTypesFoodOrderError>, InkPrim1`. The 'orderId' field contains '1'. The right side shows the 'Return value' panel displaying a JSON object with an 'Ok' key pointing to an order details object. This object includes fields like 'orderId: "1"', 'restaurantId: "1"', 'customerId: "1"', 'courierId: "9"', 'deliveryAddress: "324 Main St"', 'status: "Cooking"', 'timestamp: "1,692,079,350,328"', 'price: "50"', 'eta: "120"', and 'eta: "120"'. Below this is a 'Transactions log' panel showing the event '22/8/2023, 12:49:19 pm RestaurantService::finish_cook()'. The 'CONTRACT EVENTS' section shows 'FinishCookEvent orderId: "T"'.

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

The screenshot shows the Substrate Contracts UI interface. On the left, the sidebar displays "Astar Shibuya" and "Foodorder". The main area shows a "Caller" section with "Courier" selected. A "Message to Send" section contains the call `courierService::pickupDelivery(deliveryId: PickupDeliveryInput1): Result<Result<u64, LogicImplsTypesFoodOrderError>`. Below it, "deliveryId: PickupDeliveryInput1" is set to 1. Under "ProofSize Limit", "refTime" is 4793859072 and "ProofSize" is 125952. Under "Storage Deposit Limit", a toggle switch is off, and the "Call contract" button is visible.

Dry-run outcome

Contract call will be successful!

```
{
  Ok: '1',
}

```

GasConsumed

refTime: 4836525131 proofSize: 85761

GasRequired

refTime: 4793859072 proofSize: 125952

StorageDeposit

charge: 10.0000 nSBY

Transactions log

22/8/2023, 12:49:19 pm

RestaurantService::finish_cook()

CONTRACT EVENTS

FinishCookEvent
orderid
"T"

GENERIC EVENTS

balances: withdraw
balances: Transfer
contracts: Called
balances: Deposit

16. Check the delivery status.

The screenshot shows the Substrate Contracts UI interface. The sidebar displays "Astar Shibuya" and "Foodorder". The main area shows a "Caller" section with "Restaurant" selected. A "Message to Send" section contains the call `getService::getDeliveryFromId(deliveryId: GetDeliveryFromIdInput1): Result<Result<LogicImplsTypesDelivery, LogicImpls>`. Below it, "deliveryId: GetDeliveryFromIdInput1" is set to 1.

Outcome

Return value

```
{
  Ok: {
    orderId: '1',
    restaurantId: '1',
    customerId: '1',
    courierId: '1',
    deliveryAddress: '324 Main St',
    status: 'PickedUp',
    timestamp: '1:692,679,602,143',
  }
}
```

Transactions log

22/8/2023, 12:51:31 pm

CourierService::pickup_delivery()

CONTRACT EVENTS

PickUpDeliveryEvent
deliveryId
"T"
courierId
"T"

GENERIC EVENTS

balances: Withdraw
contracts: Called
balances: Transfer
balances: Deposit
balances: Deposit
balances: Deposit
balances: Deposit
balances: Deposit

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

The screenshot shows the Substrate Contracts UI interface. On the left, the sidebar displays 'Astar Shibuya' and a list of contracts including 'Foodorder'. The main area shows a 'Caller' section with a 'Restaurant' contract selected. A 'Message to Send' section contains the call `restaurantService::deliverOrder(orderId: DeliverOrderInput1): Result<Result<u64, LogicImplsTypesFoodOrderError>, InkP` with an input value of '1'. Below this are 'Reffime Limit' (4793859072) and 'ProofSize Limit' (125952). A 'Storage Deposit Limit' toggle is turned off. On the right, the 'Dry-run outcome' panel shows a green success message: 'Contract call will be successful!' with a JSON response object. It also displays gas consumption details like refTime (3848595929), proofSize (86937), and storageDeposit (charge: 0). The 'Transactions log' and 'CONTRACT EVENTS' sections show the execution details.

18. Check the order status.

This screenshot shows the Substrate Contracts UI after a call to `getOrderFromId`. The 'Caller' section has the 'Foodorder' contract selected. The 'Message to Send' section contains the call `getService::getOrderFromId(orderId: GetOrderFromIdInput1): Result<Result<LogicImplsTypesOrder, LogicImplsTypesFoodOrderError>, InkP` with an input value of '1'. The 'Outcome' panel on the right shows the return value as a JSON object representing an order: { Ok: { foodId: '1', restaurantId: '1', customerId: '1', courierId: '1', deliveryAddress: '324 Main St', status: 'FoodDelivered', timestamp: '1,692,679,356,328', price: '50\$', eta: '120' } }. The 'Transactions log' and 'CONTRACT EVENTS' sections show the execution details.

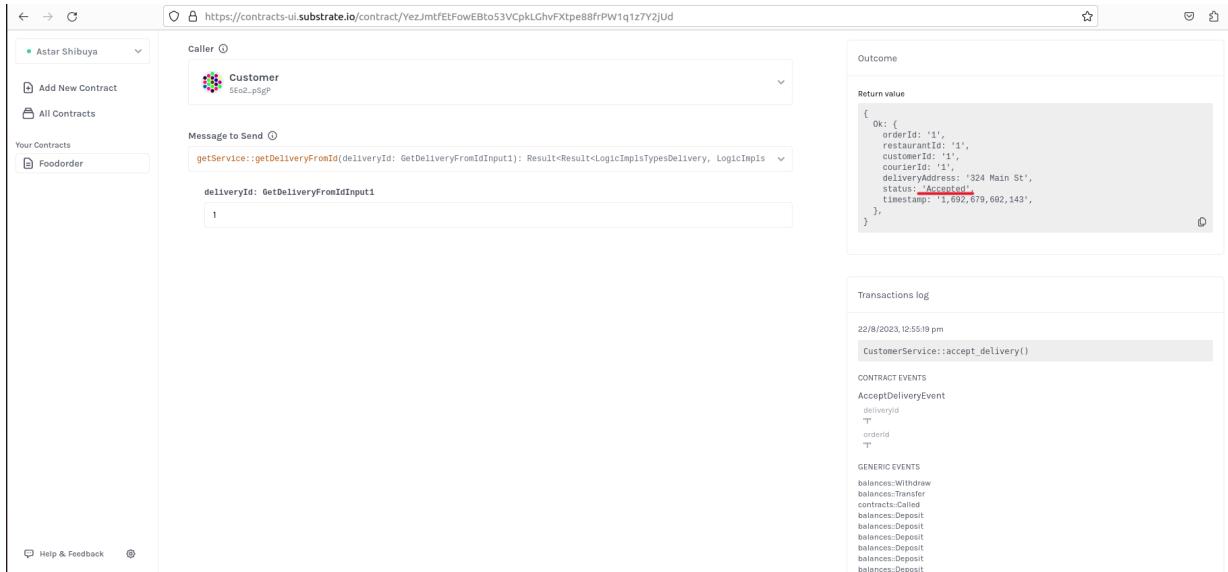
19. The customer accepts the delivery.

The screenshot shows the Substrate Contracts UI interface. On the left, there's a sidebar with 'Astar Shibuya' selected, 'Add New Contract', 'All Contracts', and a section for 'Your Contracts' with 'Foodorder' listed. The main area has a header 'https://contracts-ui.substrate.io/contract/YezJmIftElFowEBko53VCpkLGhvFXtpe88frPW1q1z7Y2jUd'. Below it, a 'Caller' dropdown shows 'Customer' (id=0...). A 'Message to Send' section contains the call: 'customerService::acceptDelivery(deliveryId: AcceptDeliveryInput1): Result<Result<u64, LogicImplsTypesFoodOrderError>,'. The input field 'deliveryId: AcceptDeliveryInput1' has a value of '1'. There are 'ReffTime Limit' (4793859072) and 'ProofSize Limit' (125952) fields. A 'Storage Deposit Limit' toggle is off. A green 'Call contract' button is at the bottom. To the right, a 'Dry-run outcome' panel shows 'Contract call will be successful!' with an execution result: '{ Ok: '1', }'. It also displays gas consumption (refTime: 4236638747, proofSize: 89527), gas required (refTime: 4793859072, proofSize: 125952), and storage deposit (charge: 0). A 'Transactions log' panel shows a single event: '22/8/2023, 12:53:08 pm RestaurantService::deliver_order()' followed by 'CONTRACT EVENTS DeliverFoodEvent orderid "1" restaurantid "1" customerid "1" courierid "1"'.

20. Check the order status.

This screenshot shows the same Substrate Contracts UI interface. The 'Message to Send' section now contains the call: 'getService::getOrderFromId(orderId: GetOrderFromIdInput1): Result<Result<LogicImplsTypesOrder, LogicImplsTypesFoodOrderError>,'. The input field 'orderId: GetOrderFromIdInput1' has a value of '1'. The right side shows the 'Outcome' panel with a 'Return value' object: '{ Ok: { foodId: '1', restaurantId: '1', customerId: '1', courierId: '1', deliveryAddress: '324 Main St', status: 'DeliveryAccepted', timestamp: '1,692,679,350,328', price: '50\$', eta: '120', } }'. The 'Transactions log' panel shows '22/8/2023, 12:55:19 pm CustomerService::accept_delivery()' followed by 'CONTRACT EVENTS AcceptDeliveryEvent deliveryId "1" orderId "1"'.

21. Check the delivery status.



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/>.

The screenshot shows the Subscan Shibuya Testnet interface. At the top, there is a navigation bar with links for Home, Blockchain, EVM, WASM, Governance, Tools, TESTNET (highlighted), and Shibuya. Below the navigation bar, a search bar contains the address "ZkedUUC36pznjiVPUtDDnijU3jQokAmxxSnpr3DnegGM893". A "Search" button is to the right of the search bar. The main content area displays information about a WASM Contract. It shows the account "ZkedUU...gGM893" and the WASM Contract address "ZkedUUC36pznjiVPUtDDnijU3jQokAmxxSnpr3DnegGM893" (underlined in red). Below this, it lists the owner as "ZvALqKqv5HftVhgH4ze7oPNf8uMmktY3TZvY3jqBqZsUuC", code hash as "0x9ce71f....b88a65d0", and deposit as "0 SBY". There are tabs for WASM Transactions, Events, Timeline, and Contract (which is selected). A note below the tabs states: "Since the wasm contract bytecode compiled on different machines and operating systems differs, we use docker to build WASM contracts deterministically. Users can use various machines and operating systems to generate the same WASM bytecode. We now provide the following docker image to compile. This ensures that Subscan is consistent with the contract deployer compilation environment." A "Contract Address" input field contains "ZkedUUC36pznjiVPUtDDnijU3jQokAmxxSnpr3DnegGM893" and a "Verify Other Contract" button is to its right.

Interacting with the smart contract

Click the button “Add an existing contract”

The screenshot shows a modal dialog box titled "add an existing contract". It contains fields for "contract address" (containing "ZkedUUC36pznjiVPUtDDnijU3jQokAmxxSnpr3DnegGM893"), "contract name" (containing "FoodOrder"), and "contract abi" (with a placeholder message: "click to select or drag and drop a JSON file"). A "Save" button is located at the bottom right of the dialog.

Type the contract address to the relevant input and upload [the ABI JSON file](#) in the contract abi box

add an existing contract ×

 FOODORDER ZkedUUC3...	
contract address ZkedUUC36pznjiVPUtDDnijU3jQokAmxxSnpr3DnegGM893	
contract name FoodOrder	
contract abi Constructors (1) ▾ Messages (31) ▾	remove abi

 Save

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

The screenshot shows the Substrate UI interface for the Shibuya Testnet. At the top, there is a blue header bar with the network name "Shibuya Testnet" and account details "shibuya/105 #4,404,402". To the right of the header are three navigation items: "Accounts", "Network", and "Governance", each with a dropdown arrow. Below the header, a sidebar on the left lists "Contracts" and "addresses", with "0" addresses shown. The main content area is titled "Contracts" and contains two entries: "CONTRACT1" and "FOODORDER", each with a message count of "Messages (31)". Below this, a section titled "code hashes" displays the message "No code hashes available".

Shibuya Testnet
shibuya/105 #4,404,402

Accounts Network Governance

Contracts

addresses 0

contracts

CONTRACT1 Messages (31)

FOODORDER Messages (31)

code hashes

No code hashes available

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) ▲ <ul style="list-style-type: none">▶ exec <code>courierService::pickupDelivery (deliveryId: PickupDeliveryInput1): Result<Result<u64, LogicImplsTypesFoodOrderError>, InkPrimitivesLangError></code>   No documentation provided▶ exec <code>customerService::acceptDelivery (deliveryId: AcceptDeliveryInput1): Result<Result<u64, LogicImplsTypesFoodOrderError>, InkPrimitivesLangError></code>   No documentation provided▶ exec <code>customerService::addCustomer (customerName: AddCustomerInput1, customerAddress: AddCustomerInput2, phoneNumber: AddCustomerInput3): Result<Result<u64, LogicImplsTypesFoodOrderError>, InkPrimitivesLangError></code>   No documentation provided▶ exec <code>customerService::submitOrder (foodId: SubmitOrderInput1, deliveryAddress: SubmitOrderInput2): Result<Result<u64, LogicImplsTypesFoodOrderError>, InkPrimitivesLangError></code>   No documentation provided▶ exec <code>managerService::addCourier (courierAccount: AddCourierInput1, courierName: AddCourierInput2, courierAddress: AddCourierInput3, phoneNumber: AddCourierInput4): Result<Result<u64, LogicImplsTypesFoodOrderError>, InkPrimitivesLangError></code>   No documentation provided▶ exec <code>managerService::addRestaurant (restaurantAccount: AddRestaurantInput1, restaurantName: AddRestaurantInput2, restaurantAddress: AddRestaurantInput3, phoneNumber: AddRestaurantInput4): Result<Result<u64, LogicImplsTypesFoodOrderError>, InkPrimitivesLangError></code>   No documentation provided▶ exec <code>managerService::changeFeeRate (rate: ChangeFeeRateInput1): Result<Result<Text, LogicImplsTypesFoodOrderError>, InkPrimitivesLangError></code>   No documentation provided▶ exec <code>managerService::changeManager (newAccount: ChangeManagerInput1): Result<Result<Text, LogicImplsTypesFoodOrderError>, InkPrimitivesLangError></code>   No documentation provided▶ exec <code>restaurantService::addFood (foodName: AddFoodInput1, description: AddFoodInput2, price: AddFoodInput3, eta: AddFoodInput4): Result<Result<u64, LogicImplsTypesFoodOrderError>, InkPrimitivesLangError></code>   No documentation provided

1. Add a restaurant.

call a contract

contract to use
FOODORDER

call from account
No accounts are available for selection.

message to send

```
managerService::addRestaurant (restaurantAccount: AddRestaurantInput1, restaurantName:  
AddRestaurantInput2, restaurantAddress: AddRestaurantInput3, phoneNumber: AddRestaurantInput4):  
Result<Result<u64, LogicImplsTypesFoodOrderError>, InkPrimitivesLangError>
```

restaurantAccount: AddRestaurantInput1
FOODORDER

restaurantName: AddRestaurantInput2
<any string>

restaurantAddress: AddRestaurantInput3
<any string>

phoneNumber: AddRestaurantInput4
<any string>

max reftime allowed (m)
32488

max proofsize allowed
3407872

0.400s execution time, 9.99% of block weight

read contract only, no execution

➡ Execute

The screenshot shows a user interface for interacting with a blockchain contract named 'FOODORDER'. At the top, it says 'call a contract' and 'contract to use FOODORDER'. Below this, there's a section for 'call from account' which displays a message: 'No accounts are available for selection.' Under the 'message to send' section, a JSON-like message is shown: 'managerService::addRestaurant (restaurantAccount: AddRestaurantInput1, restaurantName: AddRestaurantInput2, restaurantAddress: AddRestaurantInput3, phoneNumber: AddRestaurantInput4): Result<Result<u64, LogicImplsTypesFoodOrderError>, InkPrimitivesLangError>'. This message includes four input parameters: 'restaurantAccount' (set to 'FOODORDER'), 'restaurantName' (set to '<any string>'), 'restaurantAddress' (set to '<any string>'), and 'phoneNumber' (set to '<any string>'). There are also configuration fields: 'max reftime allowed (m)' set to '32488', 'max proofsize allowed' set to '3407872', and a note about execution time: '0.400s execution time, 9.99% of block weight'. At the bottom, there's a toggle switch labeled 'read contract only, no execution' which is currently turned on (indicated by a checked checkbox). A large blue button at the bottom right is labeled '➡ Execute'.

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:

call a contract

contract to use
FOODORDER

call from account
ACCOUNT 1 (SUBWALLET-JS)

transferrable 0.0000 sby
ZPJU7ALjd... ▾

message to send
`getService::getRestaurantAll (from: GetRestaurantAllInput1, to: GetRestaurantAllInput2): Result<Result<Vec<LogicImplsTypesRestaurant>, LogicImplsTypesFoodOrderError>, InkPrimitivesLangError>`

from: GetRestaurantAllInput1
0

to: GetRestaurantAllInput2
10

max reftime allowed (m)
1

max proofsize allowed
1000000

max read gas

0.400s execution time, 9.99% of block weight

Call results ▾

`getservice::getrestaurantall (from: 0, to: 10): result<result<vec<logicimplstypesrestauran... 0/11/2023 10:44:03 am`
`logicimplstypesfoodordererror>, inkprimitiveslangerror`
`{Ok: {{restaurantAccount: ZWexd44p3IVqbn5JDDXtcZvSULYqemq5423r2siVERab9b restaurantName: name1 restaurantAddress: 123 main st phoneNumber: 123456789 }}}}`

→ Read

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](#).

The screenshot shows the Shibuya wallet interface. On the left is a sidebar with the following menu items:

- Assets (highlighted)
- Dashboard
- dApp Staking
- NFT
- Ecosystem
- Forum

The main content area is titled "Assets" and displays the "Astar Native Account". It lists two accounts:

Account	Address	Balance
Account 1 (subwallet-js)	ZPJu7A.....Yrakso	0 USD
SBY	Shibuya	10 SBY (Transferable) 10 SBY

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

3. Add a customer.

contract to use
FOODORDER

ZkedUUC3...

call from account
ACCOUNT 1 (SUBWALLET-JS)

transferrable 10.0000 SBY
ZPJU7ALjd... ▾

message to send
customerService::addCustomer (customerName: AddCustomerInput1, customerAddress: AddCustomerInput2, phoneNumber: AddCustomerInput3): Result<Result<u64, LogicImplsTypesFoodOrderError>, InkPrimitivesLangError> ⚒

customerName: AddCustomerInput1
Customer2

customerAddress: AddCustomerInput2
789 Broadway Way

phoneNumber: AddCustomerInput3
0987654321

max reftime allowed (m)
32488

max proofsize allowed
3407872

.400s execution time, 9.99% of block weight

read contract only, no execution

 Execute

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

authorize transaction



contracts.call
Makes a call to an account, optionally transferring some balance
Fees of 2.1102 milli SBY will be applied to the submission

sending from my account
ACCOUNT 1 (SUBWALLET-JS) ZPJU7ALjd...

The details of the transaction including the type, the description (as available from the chain metadata) as well as any parameters and fee estimations (as available) for the specific type of call.

do not include a tip for the block author

The sending account that will be used to send this transaction. Any applicable fees will be paid by this account.

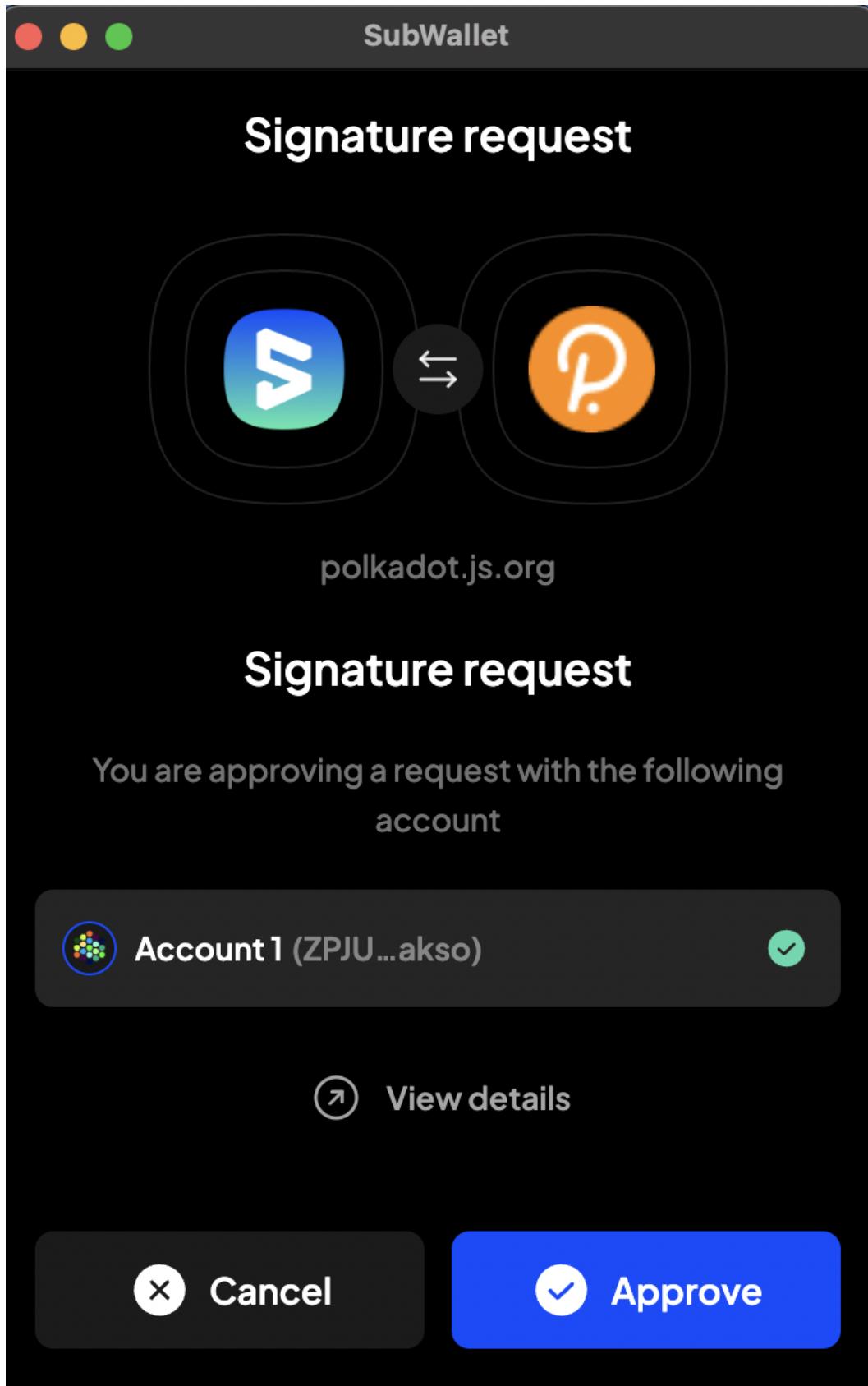
call hash
0x4a7acae4c61f73dec85e7d83870445b4df6b229101a0e19cf318c9e0c8e4821a 

Adding an optional tip to the transaction could allow for higher priority, especially when the chain is busy.

The call hash as calculated for this transaction

sign and submit

 Sign and Submit



contract to use
FOODORDER

ZkedUUC3...

call from account
ACCOUNT 1 (SUBWALLET-JS)

transferrable 9.9978 sBV
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>, inkprimitivestlangerror>`

customer1 customerAddress: 4534 main st phoneNumber: 654321987 } { customerAccount: ZPJU7ALjdVXfyp wTAVde77m8tcUTLyXv2PJ7Nqtk1Yrakso customerName: Customer2 customerAddress: 789 Broadway Way ph oneNumber: 0987654321 }] }

 Read

4. List food items

contract to use
FOODORDER

ZkedUUC3...

call from account
ACCOUNT 1 (SUBWALLET-JS)

transferrable 9.9978 SBY
ZPJU7ALjd... ▾

message to send
`getService::getFoodAll (from: GetFoodAllInput1, to: GetFoodAllInput2): Result<Result<Vec<LogicImplsTypesFood>, LogicImplsTypesFoodOrderError>, InkPrimitivesLangError>`

from: GetFoodAllInput1
0

to: GetFoodAllInput2
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::getfoodall (from: 0, to: 10): result<result<vec<logicimplstypesfood>, logicimplstypesfoodordererror, inkprimitiveslangerror>, {ok: [{foodname: food1, restaurantid: 1, description: food1 price: 1,000 eta: 2 timestamp: 1,691,776,038, 345 }]} }` 8/11/2023 11:01:58 am

X

Read

5. Consumer submits a food order

contract to use
FOODORDER ZkedUUC3...

call from account ACCOUNT 1 (SUBWALLET-JS) transferrable 9.9978 SBY ZPJU7ALjd... ▾

message to send
customerService::submitOrder (foodId: SubmitOrderInput1, deliveryAddress: SubmitOrderInput2):
 Result<Result<u64, LogicImplsTypesFoodOrderError>, InkPrimitivesLangError> ⚡

foodId: SubmitOrderInput1
1

deliveryAddress: SubmitOrderInput2
345 captain lane, CA

value
0.0000000000000001 SBY

max reftime allowed (m)
32488

max proofsize allowed
3407872

400s execution time, 9.99% of block weight

read contract only, no execution

 Execute

authorize transaction

▼ contracts.call

Makes a call to an account, optionally transferring some balance
Fees of 2.0302 milli SBY will be applied to the submission

The details of the transaction including the type, the description (as available from the chain metadata) as well as any parameters and fee estimations (as available) for the specific type of call.

sending from my account
ACCOUNT 1 (SUBWALLET-JS) ZPJU7ALjd...

do not include a tip for the block author

The sending account that will be used to send this transaction. Any applicable fees will be paid by this account.

Adding an optional tip to the transaction could allow for higher priority, especially when the chain is busy.

call hash

0x4579f1b008eeab3b126fadccb97c3487c3eba4b94678be9d871148e5738faef



The call hash as calculated for this transaction

sign and submit

 Sign and Submit

6. Confirm a order

contract to use
FOODORDER ZkedUUC36...

call from account
ACCOUNT 1 (SUBWALLET-JS) transferrable 15.3566 sBY ZWexd44p3r... ▾

message to send
restaurantService::confirmOrder (orderId: ConfirmOrderInput1, eta: ConfirmOrderInput2): Result<Result<u64, LogicImplsTypesFoodOrderError>, InkPrimitivesLangError> ▾

orderId: ConfirmOrderInput1
1

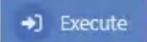
eta: ConfirmOrderInput2
200

max reftime allowed (m)
32488

max proofsize allowed
3407872

0.400s execution time, 9.99% of block weight

read contract only, no execution

 Execute

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

contract to use
FOODORDER

ZkedUUC36...

call from account
ACCOUNT 1 (SUBWALLET-JS)

transferrable 15.3547 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

.400s execution time, 9.99% of block weight

all results ▲

```
getservice::getorderall (from: 0, to: 10): result<result<vec<logicimplstypesorder>, logicimplstypesfoodordererror>, inkprimitiveslangerror>
{ Ok: { Ok: [ { foodId: 1 restaurantId: 1 customerId: 1 courierId: 0 deliveryAddress: 123 main st status: OrderConfirmed timestamp: 1,691,776,638,103 price: 1,000 eta: 200 } { foodId: 1 restaurantId: 1 customerId: 2 courierId: 0 deliveryAddress: 345 captain lane, CA status: OrderSubmitted timestamp: 1,691,777,304,084 price: 1,000 eta: 0 } ] } }
```

Read

7. Finish cooking

After food is cooked, you can call finish_cook.

contract to use
FOODORDER

ZkedUUC36...

call from account
ACCOUNT 1 (SUBWALLET-JS)

transferrable 15.3547 sBY
ZWexd44p3r... ▾

message to send
restaurantService::finishCook (orderId: FinishCookInput1): Result<Result<u64, LogicImplsTypesFoodOrderError>, InkPrimitivesLangError> ⚒

orderId: FinishCookInput1
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

And check the order status.

call from account
ACCOUNT 1 (SUBWALLET-JS)

transferrable 15.3530 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>, logicimplstypesfoodordererror>, inkprimitiveslangerror>
{ Ok: { Ok: [ { foodId: 1 restaurantId: 1 customerId: 1 courierId: 0 deliveryAddress: 123 main st status: FoodPrepared timestamp: 1,691,776,638,103 price: 1,000 eta: 200 } { foodId: 1 restaurantId: 1 customerId: 2 courierId: 0 deliveryAddress: 345 captain lane, CA status: OrderSubmitted timestamp: 1,691,777,304,084 price: 1,000 eta: 0 } ] } }
```

Read

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

call from account
ACCOUNT 1 (SUBWALLET-JS)

transferrable 15.3530 SBY
ZWexd44p3r... ▾

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>
{ Ok: { Ok: [ { orderId: 1 restaurantId: 1 customerId: 1 courierId: 0 deliveryAddress: 123 main st status: Waiting timestamp: 1,691,777,574,052 } ] } }
```

8/12/2023 3:17:23 am X

Read

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

0.400s execution time, 9.99% of block weight

read contract only, no execution



contract to use
FOODORDER

ZkedUUC3...

call from account
ACCOUNT 1 (SUBWALLET-JS)

transferrable 9.9958 sby
ZPJU7ALjd... ▾

message to send

```
getService::getCourierAll (from: GetCourierAllInput1, to: GetCourierAllInput2):  
Result<Result<Vec<LogicImplsTypesCourier>, LogicImplsTypesFoodOrderError, InkPrimitivesLangError>
```

from: GetCourierAllInput1
0

to: GetCourierAllInput2
10

max reftime allowed (m)
1

max read gas

max proofsize allowed
1000000

400s execution time, 9.99% of block weight

all results ▲

```
getservice::getcourierall (from: 0, to: 10): result<result<vec<logicimplstypescourier>, 8/11/2023 11:20:32 am  
logicimplstypesfoodordererror>, inkprimitivelangerror  
{ Ok. [{ count: Account: ZvALQHQV5Hvigh4ZcOPN5f8uMmkY3TZvY3jqBgZsUuC courierName: courier1 courierAddress: 234 main st phoneNumber: 9876542123 } ] }
```

  Read

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

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 proofsize allowed
1000000

0.400s execution time, 9.99% of block weight

Call results ▾

```
getservice::getdeliveryall (from: 0, to: 10): result<result<vec<logicimplstypesdelivery>, logicimplstypesfoodordererror>, inkprimitiveslangerror>
{ Ok: { Ok: [{ orderId: 1 restaurantId: 1 customerId: 1 courierId: 1 deliveryAddress: 123 main st status: PickedUp time stamp: 1,691,777,574,052 }]} }
```

8/12/2023 3:23:15 am



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.

call a contract

contract to use
FOODORDER

call from account
ACCOUNT 1 (SUBWALLET-JS)

message to send
`restaurantService::deliverOrder (orderId: DeliverOrderInput1): Result<Result<u64, LogicImplsTypesFoodOrderError>, InkPrimitivesLangError>`

orderid: DeliverOrderInput1
1

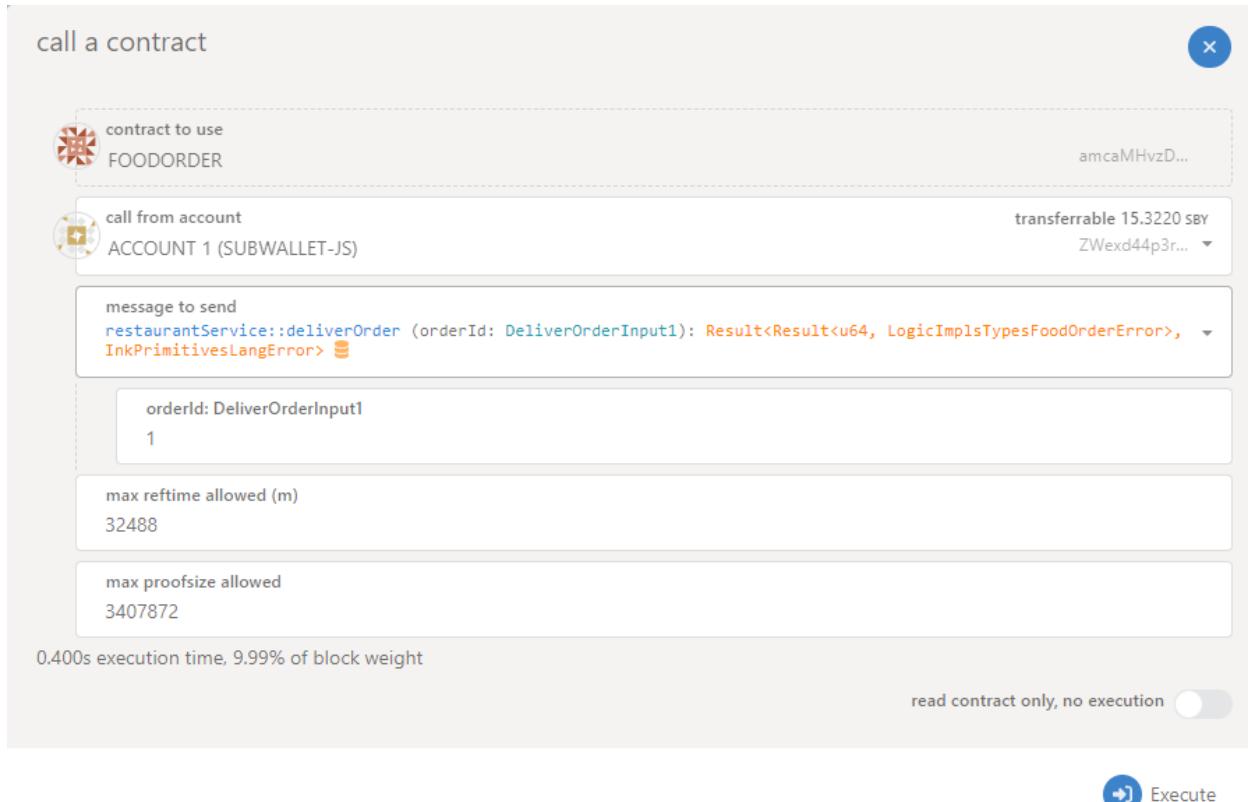
max reftime allowed (m)
32488

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

call from account
ACCOUNT 1 (SUBWALLET-JS)

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 proofsize allowed
1000000

0.400s execution time, 9.99% of block weight

Call results

`getservice::getorderall (from: 0, to: 10): result<result<vec<logicimplstypesorder>, logicimplstypesfoodordererror>, inkprimitiveslangerror>`

{Ok: {Ok: {foodId: 1 restaurantId: 1 customerId: 1 courierId: 1 deliveryAddress: asfasdfsadf status: DeliveryAccepted timestamp: 1,691,783,334,608 price: 1,000 eta: 100 } } }

8/19/2023 4:32:06 am

Read

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

✓ contracts.call inblock ✕

system.ExtrinsicSuccess
balances.Withdraw
balances.Transfer (x2)
contracts.ContractEmitted
contracts.Called
balances.Deposit (x2)
transactionPayment.TransactionFeePaid
extrinsic event

call a contract

contract to use
FOODORDER

call from account
ACCOUNT 1 (SUBWALLET-JS)

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 proofsize allowed
1000000

0.400s execution time, 9.99% of block weight

Call results ▲

```
getservice::getorderall (from: 0, to: 10): result<result<vec<logicimplstypesorder>, logicimplstypesfoodordererror>, inkprimitiveslangerror>
{ Ok: { Ok: [{ foodId: 1 restaurantId: 1 customerId: 1 courierId: 1 deliveryAddress: asfasdfsadf status: FoodDelivered timestamp: 1,691,783,334,608 price: 1,000 eta: 100 } ] } }
```

max read gas

Read