Testing with Typescript

Testing is an integral part of any piece of software, and especially important for any blockchain application. In this page we will cover how to interact with your Noir contracts in a testing environment to write automated tests for your apps.

We will be using typescript to write our tests, and rely on the aztec.js client library to interact with a local Aztec network, along with the accounts package for setting up test accounts. We will use jest as a testing library, though feel free to use whatever you work with. Configuring the nodejs testing framework is out of scope for this guide.

A simple example

Let's start with a simple example for a test using the Sandbox . We will create two accounts and deploy a token contract in a setup step, and then issue a transfer from one user to another.

sandbox-example describe ('token contract' ,

( )

=>

{ let pxe :

PXE ; let owner : AccountWallet ; let recipient : AccountWallet ; let token : TokenContract ;

beforeEach (async

( )

=>

{ pxe =

createPXEClient (PXE_URL) ; owner =

await

createAccount (pxe) ; recipient =

await

createAccount (pxe) ; token =

await TokenContract . deploy (owner , owner . getCompleteAddress ( ) ,

'TokenName' ,

'TokenSymbol' ,

18) . send ( ) . deployed ( ) ; } ,

60_000) ;

it ('increases recipient funds on mint' ,

async

( )

=>

{ const recipientAddress = recipient . getAddress ( ) ; expect (await token . methods . balance_of_private (recipientAddress) . view ( )) . toEqual (0n) ;

const mintAmount =

20n ; const secret = Fr . random ( ) ; const secretHash =

computeMessageSecretHash (secret) ; const receipt =

await token . methods . mint_private (mintAmount , secretHash) . send ( ) . wait ( ) ;

const storageSlot =

new

Fr (5) ;

// The storage slot of pending_shields is 5. const noteTypeId =

new

Fr (84114971101151129711410111011678111116101n) ;

// TransparentNote

const note =

new

Note ([new

Fr (mintAmount) , secretHash]) ; const extendedNote =

new

ExtendedNote (note , recipientAddress , token . address , storageSlot , noteTypeId , receipt . txHash ,) ; await pxe . addNote (extendedNote) ;

await token . methods . redeem_shield (recipientAddress , mintAmount , secret) . send ( ) . wait ( ) ; expect (await token . methods . balance_of_private (recipientAddress) . view ( )) . toEqual (20n) ; } ,

30_000) ; }) ;Source code: yarn-project/end-to-end/src/guides/dapp_testing.test.ts#L28-L71 This test sets up the environment by creating a client to the Private Execution Environment (PXE) running on the Sandbox on port 8080. It then creates two new accounts, dubbedowner andrecipient . Last, it deploys an instance of theToken contract , minting an initial 100 tokens to the owner.

Once we have this setup, the test itself is simple. We check the balance of therecipient user to ensure it has no tokens, send and await a deployment transaction, and then check the balance again to ensure it was increased. Note that all numeric values are represented asnative bigints to avoid loss of precision.

info We are using theToken contract's typescript interface. Follow thetypescript interface section to get type-safe methods for deploying and interacting with the token contract. To run the test, first make sure the Sandbox is running on port 8080, and thenrun your tests using jest . Your test should pass, and you should see the following output in the Sandbox logs, where each chunk corresponds to a transaction. Note how this test run has a total of four transactions: two for deploying the account contracts for theowner andrecipient , another for deploying the token contract, and a last one for actually executing the transfer.

pxe_service Registered account 0x061c94e053745973521de1826db5a1ee24af60a3c203c294570a35bd5afa3286 pxe_service Added contract SchnorrAccount at 0x061c94e053745973521de1826db5a1ee24af60a3c203c294570a35bd5afa3286 node Simulating tx 09023befa12d01db063235ef6d611ed1c7ba4625dac9d14cc0f1ff4dd8a00264 node Simulated tx 09023befa12d01db063235ef6d611ed1c7ba4625dac9d14cc0f1ff4dd8a00264 succeeds pxe_service Executed local simulation for 09023befa12d01db063235ef6d611ed1c7ba4625dac9d14cc0f1ff4dd8a00264 pxe_service Sending transaction 09023befa12d01db063235ef6d611ed1c7ba4625dac9d14cc0f1ff4dd8a00264 node Received tx 09023befa12d01db063235ef6d611ed1c7ba4625dac9d14cc0f1ff4dd8a00264 sequencer Retrieved 1 txs from P2P pool sequencer Building block 1 with 1 transactions sequencer Submitted rollup block 1 with 1 transactions

pxe_service Registered account 0x109e72d06371d98cef1a10ec137e01139edb64b3b399c1b761d12d06de15af3c pxe_service Added contract SchnorrAccount at 0x109e72d06371d98cef1a10ec137e01139edb64b3b399c1b761d12d06de15af3c node Simulating tx 179d347ff45fc8da90155c38c7fd4358737fc7447b1f2ea2b3ff1fbc9dfb3d47 node Simulated tx 179d347ff45fc8da90155c38c7fd4358737fc7447b1f2ea2b3ff1fbc9dfb3d47 succeeds pxe_service Executed local simulation for 179d347ff45fc8da90155c38c7fd4358737fc7447b1f2ea2b3ff1fbc9dfb3d47 pxe_service Sending transaction 179d347ff45fc8da90155c38c7fd4358737fc7447b1f2ea2b3ff1fbc9dfb3d47 node Received tx 179d347ff45fc8da90155c38c7fd4358737fc7447b1f2ea2b3ff1fbc9dfb3d47 sequencer Retrieved 1 txs from P2P pool sequencer Building block 2 with 1 transactions sequencer Submitted rollup block 2 with 1 transactions

pxe_service Added contract Token at 0x1345499a3f8325d614fa1d49cc2a6f21211d608c74728439076943f92340b936 node Simulating tx 0298295963669a4a1ccaddb40d78722c00136aad196b85306d63e4885799b1d8 node Simulated tx 0298295963669a4a1ccaddb40d78722c00136aad196b85306d63e4885799b1d8 succeeds pxe_service Executed local simulation for 0298295963669a4a1ccaddb40d78722c00136aad196b85306d63e4885799b1d8 pxe_service Sending transaction 0298295963669a4a1ccaddb40d78722c00136aad196b85306d63e4885799b1d8 node Received tx 0298295963669a4a1ccaddb40d78722c00136aad196b85306d63e4885799b1d8 sequencer Retrieved 1 txs from P2P pool sequencer Building block 3 with 1 transactions sequencer Submitted rollup block 3 with 1 transactions

node Simulating tx 085665fbbad776a727cb92c4b62fbe2f57c83dfbccd191852e3c17efc12fdd4b node Simulated tx 085665fbbad776a727cb92c4b62fbe2f57c83dfbccd191852e3c17efc12fdd4b succeeds pxe_service Executed local simulation for 085665fbbad776a727cb92c4b62fbe2f57c83dfbccd191852e3c17efc12fdd4b pxe_service Sending transaction 085665fbbad776a727cb92c4b62fbe2f57c83dfbccd191852e3c17efc12fdd4b node Received tx 085665fbbad776a727cb92c4b62fbe2f57c83dfbccd191852e3c17efc12fdd4b sequencer Retrieved 1 txs from P2P pool sequencer Building block 4 with 1 transactions sequencer Submitted rollup block 4 with 1 transactions

node Simulating tx 2755e9f5ad308fd135f606daf6208f5d711a3a6de6e4630d15d269af59f03e1c node Simulated tx 2755e9f5ad308fd135f606daf6208f5d711a3a6de6e4630d15d269af59f03e1c succeeds pxe_service Executed local simulation for 2755e9f5ad308fd135f606daf6208f5d711a3a6de6e4630d15d269af59f03e1c pxe_service Sending transaction 2755e9f5ad308fd135f606daf6208f5d711a3a6de6e4630d15d269af59f03e1c node Received tx 2755e9f5ad308fd135f606daf6208f5d711a3a6de6e4630d15d269af59f03e1c sequencer Retrieved 1 txs from P2P pool sequencer Building block 5 with 1 transactions sequencer Submitted rollup block 5 with 1 transactions

Using Sandbox initial accounts

Instead of creating new accounts in our test suite, we can use the ones already initialized by the Sandbox upon startup. This can provide a speed boost to your tests setup. However, bear in mind that you may accidentally introduce an interdependency across test suites by reusing the same accounts.

use-existing-wallets pxe =

createPXEClient (PXE_URL) ; [owner , recipient]

=

await

getDeployedTestAccountsWallets (pxe) ; token =

await TokenContract . deploy (owner , owner . getCompleteAddress ( ) ,

'TokenName' ,

'TokenSymbol' ,

18) . send ( ) . deployed ( ) ; Source code: yarn-project/end-to-end/src/guides/dapp_testing.test.ts#L80-L86

Using debug options

You can use thedebug option in thewait method to get more information about the effects of the transaction. At the time of writing, this includes information about new note hashes added to the note hash tree, new nullifiers, public data writes, new L2 to L1 messages, new contract information and newly visible notes.

This debug information will be populated in the transaction receipt. You can log it to the console or use it to make assertions about the transaction.

If a note doesn't appear when you expect it to, check the visible notes returned by the debug options. See the following example for reference on how it's done in the token contract tests.

debug const receiptClaim =

await txClaim . wait ({ debug :

true

}) ; Source code: yarn-project/end-to-end/src/e2e_token_contract.test.ts#L283-L285 If the note appears in the visible notes and it contains the expected values there is probably an issue with how you fetch the notes. Check that the note getter (or note viewer) parameters are set correctly. If the note doesn't appear, ensure that you have emitted the corresponding encrypted log (usually by passing in abroadcast = true param to thecreate_note function). You can also check the Sandbox logs to see if theemitEncryptedLog was emitted. Run `export DEBUG="aztec:*" before spinning up sandbox to see all the logs.

For debugging and logging in Aztec contracts, see this page .

Assertions

We will now see how to useaztec.js to write assertions about transaction statuses, about chain state both public and private, and about logs.

Transactions

In the example above we usedcontract.methods.method().send().wait() to create a function call for a contract, send it, and await it to be mined successfully. But what if we want to assert failing scenarios?

A private call fails

We can check that a call to a private function would fail by simulating it locally and expecting a rejection. Remember that all private function calls are only executed locally in order to preserve privacy. As an example, we can try transferring more tokens than we have, which will fail an assertion with theBalance too low error message.

local-tx-fails const call = token . methods . transfer (owner . getAddress ( ) , recipient . getAddress ( ) ,

200n ,

0) ; await

expect (call . simulate ( )) . rejects . toThrowError (/ Balance too low /)Source code: yarn-project/end-to-end/src/guides/dapp_testing.test.ts#L221-L224 Under the hood, thesend() method executes a simulation, so we can just call the usualsend().wait() to catch the same failure.

local-tx-fails-send const call = token . methods . transfer (owner . getAddress ( ) , recipient . getAddress ( ) ,

200n ,

0) ; await

expect (call . send ( ) . wait ( )) . rejects . toThrowError (/ Balance too low /)Source code: yarn-project/end-to-end/src/guides/dapp_testing.test.ts#L228-L231

A transaction is dropped

We can have private transactions that work fine locally, but are dropped by the sequencer when tried to be included due to a double-spend. In this example, we simulate two different transfers that would succeed individually, but not when both are tried to be mined. Here we need tosend() the transaction andwait() for it to be mined.

tx-dropped const call1 = token . methods . transfer (owner . getAddress ( ) , recipient . getAddress ( ) ,

80n ,

0) ; const call2 = token . methods . transfer (owner . getAddress ( ) , recipient . getAddress ( ) ,

50n ,

0) ;

await call1 . simulate ( ) ; await call2 . simulate ( ) ;

await call1 . send ( ) . wait ( ) ; await

expect (call2 . send ( ) . wait ( )) . rejects . toThrowError (/ dropped /)Source code: yarn-project/end-to-end/src/guides/dapp_testing.test.ts#L235-L244

A public call fails locally

Public function calls can be caught failing locally similar to how we catch private function calls. For this example, we use aTokenContract instead of a private one.

info Keep in mind that public function calls behave as in EVM blockchains, in that they are executed by the sequencer and not locally. Local simulation helps alert the user of a potential failure, but the actual execution path of a public function call will depend on when it gets mined. local-pub-fails const call = token . methods . transfer_public (owner . getAddress ( ) , recipient . getAddress ( ) ,

1000n ,

0) ; await

expect (call . simulate ( )) . rejects . toThrowError (U128_UNDERFLOW_ERROR) Source code: yarn-project/end-to-end/src/guides/dapp_testing.test.ts#L248-L251

A public call fails on the sequencer

We can ignore a local simulation error for a public function via theskipPublicSimulation . This will submit a failing call to the sequencer, who will include the transaction, but without any side effects from our application logic.

pub-reverted const call = token . methods . transfer_public (owner . getAddress ( ) , recipient . getAddress ( ) ,

1000n ,

0) ; const receipt =

await call . send ({ skipPublicSimulation :

true

}) . wait ( ) ; expect (receipt . status) . toEqual (TxStatus . MINED) ; const ownerPublicBalanceSlot = cheats . aztec . computeSlotInMap (6n , owner . getAddress ( )) ; const balance =

await pxe . getPublicStorageAt (token . address , ownerPublicBalanceSlot) ; expect (balance . value) . toEqual (100n) ; Source code: yarn-project/end-to-end/src/guides/dapp_testing.test.ts#L256-L263 WARN Error processing tx 06dc87c4d64462916ea58426ffcfaf20017880b353c9ec3e0f0ee5fab3ea923f: Assertion failed: Balance too low. info Presently, the transaction is included, but no additional information is included in the block to mark it as reverted. This will change in the near future.

State

We can check private or public state directly rather than going through view-only methods, as we did in the initial example by callingtoken.methods.balance().view() . Bear in mind that directly accessing contract storage will break any kind of encapsulation.

To query storage directly, you'll need to know the slot you want to access. This can be checked in thecontract'sStorage definition directly for most data types. However, when it comes to mapping types, as in most EVM languages, we'll need to calculate the slot for a given key. To do this, we'll use theCheatCodes utility class:

calc-slot cheats = CheatCodes . create (ETHEREUM_HOST , pxe) ; // The balances mapping is defined on storage slot 3 and is indexed by user address ownerSlot = cheats . aztec . computeSlotInMap (3n , ownerAddress) ; Source code: yarn-project/end-to-end/src/guides/dapp_testing.test.ts#L178-L182

Querying private state

Private state in the Aztec Network is represented via sets ofprivate notes . In our token contract example, the balance of a user is represented as a set of unspent value notes, each with their own corresponding numeric value.

value-note-def struct

ValueNote

{ value :

Field , owner :

AztecAddress , randomness :

Field , header :

NoteHeader , } Source code: noir-projects/aztec-nr/value-note/src/value_note.nr#L10-L17 We can query the Private eXecution Environment (PXE) for all notes encrypted for a given user in a contract slot. For this example, we'll get all notes encrypted for theowner user that are stored on the token contract address and on the slot we calculated earlier. To calculate the actual balance, we extract thevalue of each note, which is the first element, and sum them up.

private-storage const notes =

await pxe . getNotes ({ owner : owner . getAddress ( ) , contractAddress : token . address , storageSlot : ownerSlot , }) ; const values = notes . map (note => note . note . items [0]) ; const balance = values . reduce ((sum , current)

=> sum + current . toBigInt ( ) ,

0n) ; expect (balance) . toEqual (100n) Source code: yarn-project/end-to-end/src/guides/dapp_testing.test.ts#L186-L195

Querying public state

Public state behaves as a key-value store, much like in the EVM. This scenario is much more straightforward, in that we can directly query the target slot and get the result back as a buffer. Note that we use theTokenContract in this example, which defines a mapping of public balances on slot 6.

public-storage await token . methods . mint_public (owner . getAddress ( ) ,

100n) . send ( ) . wait ( ) ; const ownerPublicBalanceSlot = cheats . aztec . computeSlotInMap (6n , owner . getAddress ( )) ; const balance =

await pxe . getPublicStorageAt (token . address , ownerPublicBalanceSlot) ; expect (balance . value) . toEqual (100n) ; Source code: yarn-project/end-to-end/src/guides/dapp_testing.test.ts#L199-L204

Logs

Last but not least, we can check the logs ofevents emitted by our contracts. Contracts in Aztec can emit bothencrypted andunencrypted events.

info At the time of this writing, only unencrypted events can be queried directly. Encrypted events are always assumed to be encrypted notes.

Querying unencrypted logs

We can query the PXE for the unencrypted logs emitted in the block where our transaction is mined. Note that logs need to be unrolled and formatted as strings for consumption.

unencrypted-logs const value = Fr . fromString ('ef') ;

// Only 1 bytes will make its way in there :(so no larger stuff const tx =

await testContract . methods . emit_unencrypted (value) . send ( ) . wait ( ) ; const filter =

{ fromBlock : tx . blockNumber ! , limit :

1 ,

// 1 log expected } ; const logs =

(await pxe . getUnencryptedLogs (filter)) . logs ; expect (Fr . fromBuffer (logs [0] . log . data)) . toEqual (value) ; Source code: yarn-project/end-to-end/src/guides/dapp_testing.test.ts#L208-L217

Cheats

TheCheatCodes class, which we used forcalculating the storage slot above , also includes a set of cheat methods for modifying the chain state that can be handy for testing.

Set next block timestamp

Thewarp method sets the time for next execution, both on L1 and L2. We can test this using anisTimeEqual function in aTest contract defined like the following:

is-time-equal

[aztec(public)]

fn

is_time_equal (time :

Field)

->

Field

{ assert (context . timestamp ( )

== time) ; time }Source code: noir-projects/noir-contracts/contracts/test_contract/src/main.nr#L243-L249 We can then callwarp and rely on theisTimeEqual function to check that the timestamp was properly modified.

warp const newTimestamp = Math . floor (Date . now ( )

/

1000)

+

60

*

60

*

24 ; await cheats . aztec . warp (newTimestamp) ; await testContract . methods . is_time_equal (newTimestamp) . send ( ) . wait ( ) ; Source code: yarn-project/end-to-end/src/guides/dapp_testing.test.ts#L130-L134 info Thewarp method callsevm_setNextBlockTimestamp under the hood on L1. Edit this page

Previous Deploy & Call Contracts with Typescript Next Sandbox and CLI