ENTERSOFT

MyEG

Zetrix

Interim Smart Contract Audit Report

DATE: 10 April 2024 | **PREPARED BY:** Entersoft Pte Ltd

Contents

Revision History & Version Control

Version	Date	Author(s)	Description
1.0	10 April 2024	Gurkirat Singh Tharun Betina Nishita Palaksha	Interim report

Entersoft was commissioned by MyEG to perform Smart contract code review on Zetrix. The review was conducted from 9th February 2024 to 10th April 2024 on their updated code, to ensure overall code quality, security, and correctness and that the code will work as intended.

The report is structured into two main sections:

- Executive Summary which provides a high-level overview of the audit findings.
- Technical Analysis which offers a detailed analysis of the smart contract code.

Please note that the analysis is Manual and entirely limited to the smart contract code. The information provided in this report should be used to understand the security and quality of the code, as well as its expected behavior.

Audit Scope includes:

- Dpos.js

Standards followed include:

- OWASP (partially, for instance, role validations, input validations, etc.)

1.0 Disclaimer

This is a limited audit report on our findings based on our analysis, in accordance with good industry practice as at the date of this report, in relation to (i) smart contract best coding practices and issues in the framework and algorithms based on white paper, code, the details of which are set out in this report, (Smart Contract audit). To get a full view of our analysis, you must read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us based on what it says or does not say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the disclaimer below – please make sure to read it in full.

DISCLAIMER: By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and Entersoft Australia and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives) (Entersoft) owe no duty of care towards you or any other person, nor does Entersoft make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties, or other terms of any kind except as set out in this disclaimer, and Entersoft hereby excludes all representations, warranties, conditions, and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose, and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, Entersoft hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against Entersoft, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of the use of this report, and any reliance on this report. The analysis of the Smart contract is purely based on the smart contract code shared with us alone.

2.0 Overview

2.1 Project Overview

Entersoft has meticulously audited the smart contract project from 20th December 2023 to 11th April 2024, with a primary focus on Javascript code files integral to blockchain functionality. The working of basic functionalities was tested during the review.

2.2 Scope

The audit scope covers the Zetrix smart contracts available in the GitHub private repository. The audit focused on the checklist provided for the smart contract code audit.

1. Dpos.js

2.3 Project Summary

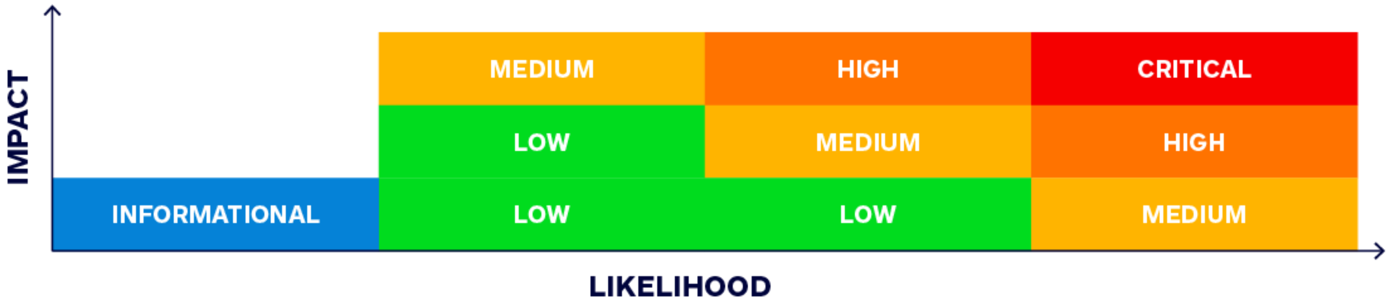
Name	Verified	Audited	Vulnerabilities
Zetrix	No	Yes	Please review Section 4

2.4 Audit Summary

Delivery Date	Method of Audit	Consultants Engaged
11th April 2024	The Manual part of a smart contract audit refers to the process of reviewing the code of a smart contract to identify security vulnerabilities, coding errors, and adherence to best practices while executing the code in a test environment. The smart contract is reviewed as is. In this we will cover functions logic vulnerabilities.	2

2.5 Security Level References

Every vulnerability in this report was assigned a severity level from the following classification table:



	MEDIUM	HIGH	CRITICAL
	LOW	MEDIUM	HIGH
INFORMATIONAL	LOW	LOW	MEDIUM

2.6 Vulnerability Summary

● Critical	● High	● Medium	● Low	● Informational
0	0	0	0	1

3.0 Executive Summary

Entersoft has conducted a dynamic smart contract audit of the Zetrix project through a comprehensive smart contract audit. The primary objective was to identify potential vulnerabilities and risks within the codebase, ensuring adherence to industry-leading standards while prioritizing security, reliability, and performance. Our focus was on prompt and efficient identification and resolution of vulnerabilities to enhance the overall robustness of the javascript smart contract.

Testing Methodology:

Our testing methodology in Javascript adhered to industry standards and best practices, integrating partially implemented OWASP and NIST SP 800 standards for encryption and signatures.

We have performed a detailed manual analysis, adherence to industry standards, and the use of a comprehensive toolset. Our approach ensured a thorough evaluation within the designated Javascript code files.

Findings and Security Posture:

Our primary focus was on Access Control Policies, Transaction Signature Validations, Time Manipulation, Default Visibility, Input Validation, Shadowing State Variables, Presence of Unused Variables, Overflow and Underflow Conditions, Assets Integrity, Errors and Exception.

Importantly, our audit process intentionally avoided reliance solely on automated tools, emphasizing a more in-depth and nuanced approach to security analysis. Conducted from December 20, 2023, to April 11, 2024, our team diligently assessed and validated the security posture of the Javascript smart contract. Entersoft has identified an informational finding, which is discussed in detail in the technical analysis section. We want to verify that this finding does not impact the security stance of the contracts, ultimately classifying it as "Secure," reflecting the absence of vulnerabilities and the robustness of the codebase against potential threats.

Result

The following table provides an overall summary of the findings and security posture of the smart contract code in scope.

- ✔ **No Security vulnerabilities were identified**
- **X Security vulnerabilities were identified**

#	Zetrix Audit Attack Vectors	Result
1	Access Control Policies	✔
2	Transaction Signature Validations	✔
3	Default Visibility	✔
4	Address Validation	✔
5	Presence of unused variables	✔
6	Assets Integrity	✔
7	Program Validations	❌
8	DOS	✔
Overall Security Posture		**Secure**

Comprehensive Analysis findings

In our rigorous smart contract audit process, we employed a multifaceted approach combining both static and dynamic analysis methodologies to comprehensively evaluate the security posture of the protocol.

Phase 1: Static Analysis

After conducting a rigorous static analysis using Sonarqube, our team meticulously examined every facet of the smart contract. We are pleased to report that no vulnerabilities were uncovered within the codebase, affirming its suitability for deployment in a secure and reliable manner within the blockchain ecosystem.

Phase 2: Dynamic Analysis

We involved a more hands-on approach to scrutinizing the intended functionality and security of the smart contracts. Further, we devised a comprehensive suite of unit tests to validate the expected behavior of the smart contracts under various scenarios.

Processes and Test Cases:

Our audit encompassed a systematic exploration of the smart contracts' functionalities, focusing on critical areas prone to vulnerabilities. We meticulously crafted test cases to assess the resilience of the contracts against potential attack vectors, including but not limited to reentrancy attacks, DOS vulnerabilities, front-running exploits, time-sensitive vulnerability, and arithmetic errors.

Throughout the static and dynamic analysis phases, our team dedicated substantial efforts to meticulously review the codebase, identify vulnerabilities, and develop robust test cases to assess the security posture of the protocol comprehensively. By combining automated analysis with manual examination and testing, we ensured a thorough evaluation of the smart contracts, ultimately enhancing the security and reliability of the protocol.

4.0 Technical Analysis

Note: The following values for “Severity” mean:

- Critical: This vulnerability poses a direct and severe threat to the funds or the main functionality of the protocol.
- High: Direct impact on the funds or the main functionality of the protocol.
- Medium: Indirect impact on the funds or the protocol's functionality.
- Low: Minimal to no impact on the funds or the protocol's main functionality.
- Informational: Suggestions related to good coding practices and gas-efficient code.

4.1 Committee apply check

<table>
<tr><td><b>Result</b></td><td><b>PASS</b></td></tr>
<tr><td><b>Function</b></td><td>Apply and Approve</td></tr>
<tr><td><b>Objective</b></td><td>Checks the process of applying for a committee role. It invokes the "apply" function with the specified parameters and verifies the result.</td></tr>
<tr><td><b>Test Case</b></td><td><pre>
// committee - sourceAddress
describe('Apply and Approve test 1', async function () {
  this.timeout(30000);

  it('committee apply check', function* () {

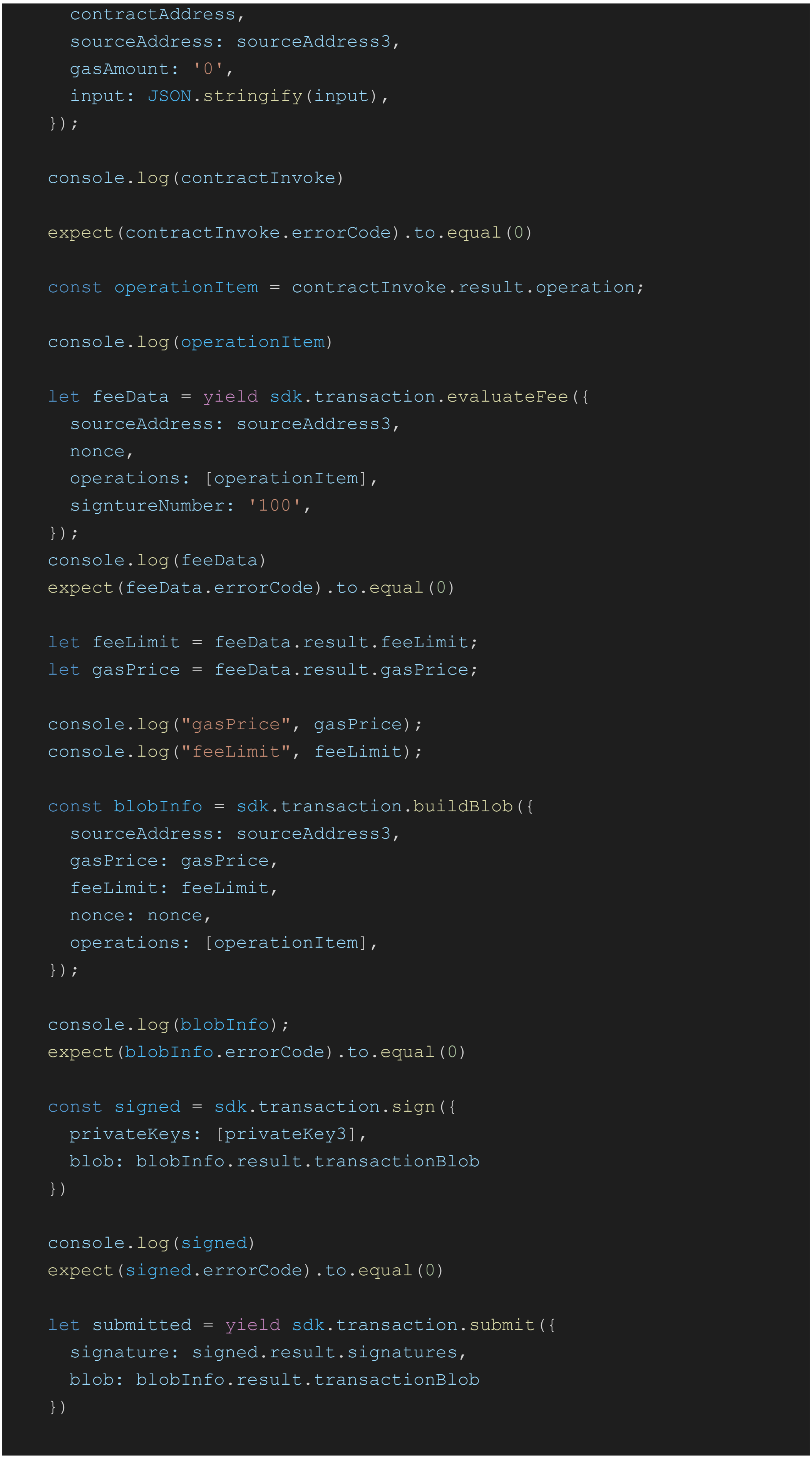
    const nonceResult = yield sdk.account.getNonce(sourceAddress3);

    expect(nonceResult.errorCode).to.equal(0)

    let nonce = nonceResult.result.nonce;
    nonce = new BigNumber(nonce).plus(1).toString(10);

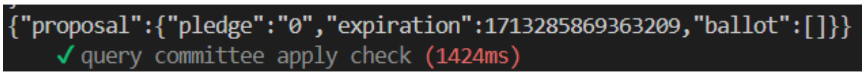
    /*
     Specify the input parameters for invoking contract
     */
    let input = {
      "method": "apply",
      "params": {
        "role": "committee",
        "ratio": 0,
        "node": "sourceAddress3"
      }
    }

    let contractInvoke = yield
sdk.operation.contractInvokeByGasOperation({
</pre></td></tr>
</table>



```
  contractAddress,
  sourceAddress: sourceAddress3,
  gasAmount: '0',
  input: JSON.stringify(input),
});

console.log(contractInvoke)

expect(contractInvoke.errorCode).to.equal(0)

const operationItem = contractInvoke.result.operation;

console.log(operationItem)

let feeData = yield sdk.transaction.evaluateFee({
  sourceAddress: sourceAddress3,
  nonce,
  operations: [operationItem],
  signtureNumber: '100',
});
console.log(feeData)
expect(feeData.errorCode).to.equal(0)

let feeLimit = feeData.result.feeLimit;
let gasPrice = feeData.result.gasPrice;

console.log("gasPrice", gasPrice);
console.log("feeLimit", feeLimit);

const blobInfo = sdk.transaction.buildBlob({
  sourceAddress: sourceAddress3,
  gasPrice: gasPrice,
  feeLimit: feeLimit,
  nonce: nonce,
  operations: [operationItem],
});

console.log(blobInfo);
expect(blobInfo.errorCode).to.equal(0)

const signed = sdk.transaction.sign({
  privateKeys: [privateKey3],
  blob: blobInfo.result.transactionBlob
})

console.log(signed)
expect(signed.errorCode).to.equal(0)

let submitted = yield sdk.transaction.submit({
  signature: signed.result.signatures,
  blob: blobInfo.result.transactionBlob
})
```

```
    console.log(submitted)
    expect(submitted.errorCode).to.equal(0)

    let info = null;
    for (let i = 0; i < 10; i++) {
      console.log("Getting the transaction history (attempt " + (i +
1).toString() + ")...")
      info = yield sdk.transaction.getInfo(submitted.result.hash)
      if (info.errorCode === 0) {
        break;
      }
      sleep(2000);
    }

    expect(info.errorCode).to.equal(0)

  });

  it('query committee apply check', async () => {

    let data = await sdk.contract.call({
      optType: 2,
      contractAddress: contractAddress,
      input: JSON.stringify({
        method: 'getProposal',
           params: {
             operate: 'apply',
             item: 'committee',
             address: 'sourceAddress3'
           }
      }),
    });
    console.log(data)
    console.log(data.result.query_rets[0].result.value);
    expect(data.errorCode).to.equal(0);

expect(data.result.query_rets[0].result.value).to.not.equal('{"proposal"
:false}');
  });


  });
```

Expected Behavior	As "proposal" is not false, the user has successfully applied to be a committee member. 

```
{"proposal":{"pledge":"0","expiration":1713285869363209,"ballot":[]}}
    ✓ query committee apply check (1424ms)
```

4.2 Committee repeated apply check

<table>
<tr><td><b>Result</b></td><td><b>PASS</b></td></tr>
<tr><td><b>Function</b></td><td>Apply and Approve</td></tr>
<tr><td><b>Objective</b></td><td>Verifies that the same account can’t apply twice for the same role. Tries applying for committee role by the same committee candidate who has applied earlier and thus expects an error.</td></tr>
<tr><td><b>Test Case</b></td><td><pre><code>// committee - sourceAddress; committee candidate - sourceAddress3

describe('Apply and Approve test 2', async function () {
  this.timeout(30000);

  it('committee repeated apply check', function* () {

    const nonceResult = yield sdk.account.getNonce(sourceAddress3);

    expect(nonceResult.errorCode).to.equal(0)

    let nonce = nonceResult.result.nonce;
    nonce = new BigNumber(nonce).plus(1).toString(10);

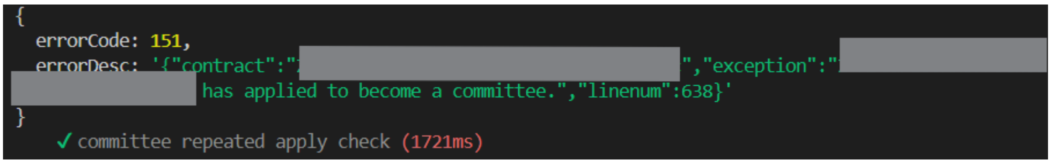
    /*
     Specify the input parameters for invoking contract
     */
    let input = {
      "method": "apply",
      "params": {
        "role": "committee",
        "ratio": 0,
        "node": "sourceAddress3"
      }
    }

    let contractInvoke = yield
sdk.operation.contractInvokeByGasOperation({
      contractAddress,
      sourceAddress: sourceAddress3,
      gasAmount: '0',
      input: JSON.stringify(input),
    });

    console.log(contractInvoke)</code></pre></td></tr>
</table>

```
    expect(contractInvoke.errorCode).to.equal(0)

    const operationItem = contractInvoke.result.operation;

    console.log(operationItem)

    let feeData = yield sdk.transaction.evaluateFee({
      sourceAddress: sourceAddress3,
      nonce,
      operations: [operationItem],
      signtureNumber: '100',
    });
    console.log(feeData)
    expect(feeData.errorCode).to.equal(151)

  });

  });
```

Expected Behavior	Same member trying to apply for the same role raises an error.

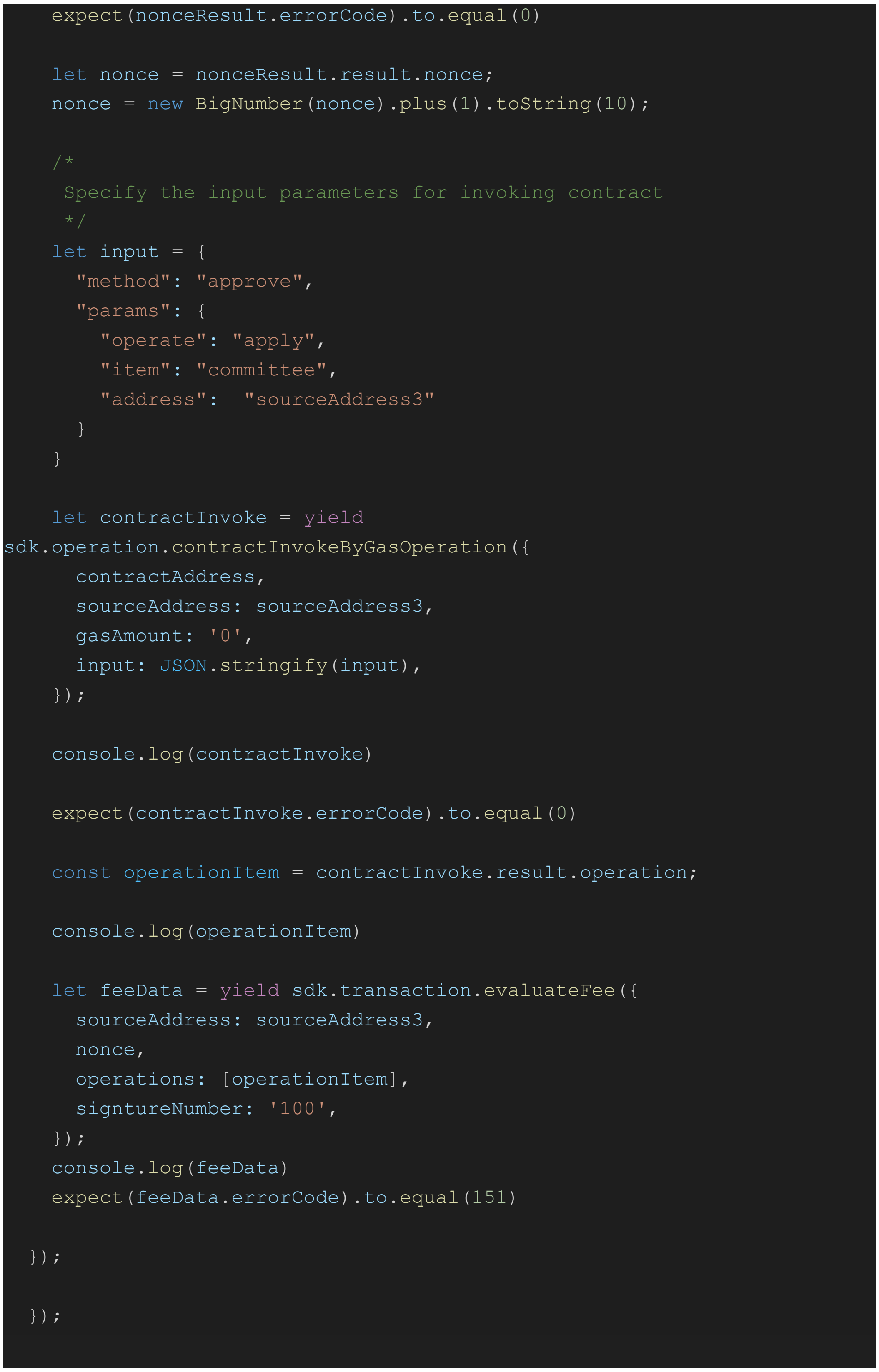


```
{
  errorCode: 151,
  errorDesc: '{"contract":"[redacted]","exception":"[redacted]
[redacted] has applied to become a committee.","linenum":638}'
}
    ✓ committee repeated apply check (1721ms)
```

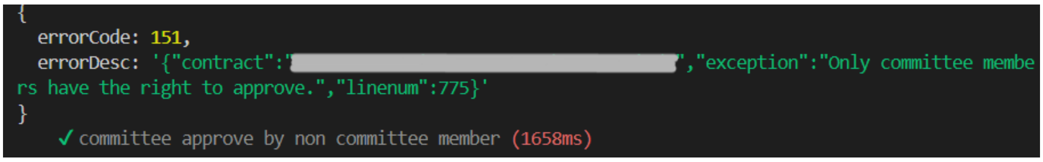
4.3 committee approve by non committee member

Result	**PASS**
Function	Apply and Approve
Objective	Verifies that a non - committee member can’t approve a committee apply proposal. Here, a non - committee member tries to approve a committee apply proposal and thus expects an error.
Test Case	

```
// committee - sourceAddress; committee candidate - sourceAddress3
describe('Apply and Approve test 3', async function () {
  this.timeout(30000);

  it('committee approve by non committee member', function* () {

    const nonceResult = yield sdk.account.getNonce(sourceAddress3);
```

<table>
<tr><td></td><td>


```
    expect(nonceResult.errorCode).to.equal(0)

    let nonce = nonceResult.result.nonce;
    nonce = new BigNumber(nonce).plus(1).toString(10);

    /*
     Specify the input parameters for invoking contract
     */
    let input = {
      "method": "approve",
      "params": {
        "operate": "apply",
        "item": "committee",
        "address":  "sourceAddress3"
      }
    }

    let contractInvoke = yield
sdk.operation.contractInvokeByGasOperation({
      contractAddress,
      sourceAddress: sourceAddress3,
      gasAmount: '0',
      input: JSON.stringify(input),
    });

    console.log(contractInvoke)

    expect(contractInvoke.errorCode).to.equal(0)

    const operationItem = contractInvoke.result.operation;

    console.log(operationItem)

    let feeData = yield sdk.transaction.evaluateFee({
      sourceAddress: sourceAddress3,
      nonce,
      operations: [operationItem],
      signtureNumber: '100',
    });
    console.log(feeData)
    expect(feeData.errorCode).to.equal(151)

  });

  });
```
</td></tr>
<tr><td><b>Expected Behavior</b></td><td>A non-committee member trying to approve a committee apply proposal raises an error.</td></tr>
</table>



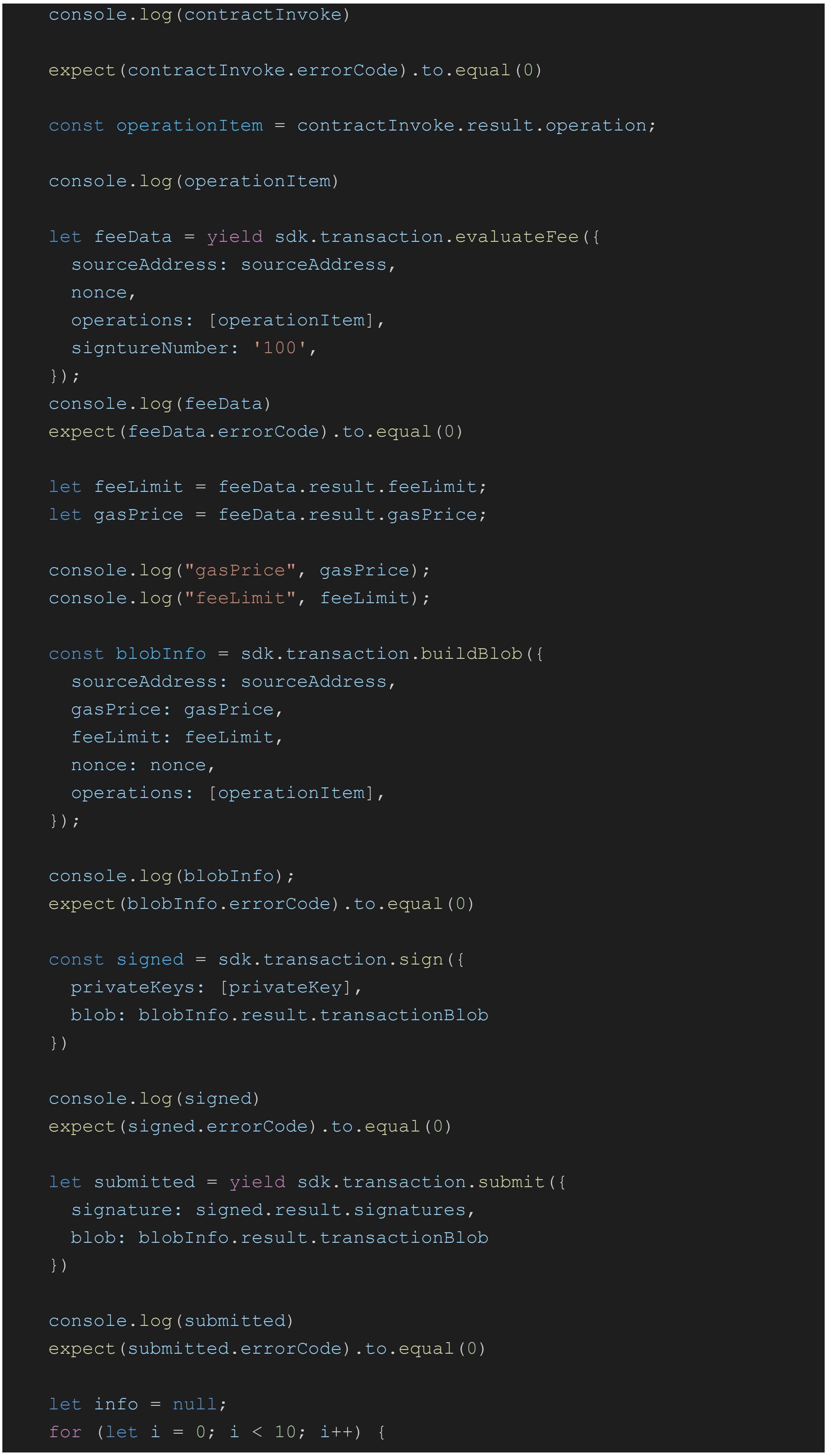
```
{
  errorCode: 151,
  errorDesc: '{"contract":"[redacted]","exception":"Only committee membe
rs have the right to approve.","linenum":775}'
}
    ✓ committee approve by non committee member (1658ms)
```

4.4 Committee approve check

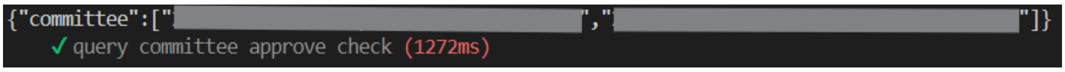
Result	**PASS**
Function	Apply and Approve
Objective	Checks the process of approving a committee role proposal. It invokes the "approve" function with the specified parameters and verifies the result.

Test Case

```
// committee - sourceAddress; committee candidate - sourceAddress3
describe('Apply and Approve test 4', async function () {
  this.timeout(30000);

  it('committee approve check', function* () {

    const nonceResult = yield sdk.account.getNonce(sourceAddress);

    expect(nonceResult.errorCode).to.equal(0)

    let nonce = nonceResult.result.nonce;
    nonce = new BigNumber(nonce).plus(1).toString(10);

    /*
     Specify the input parameters for invoking contract
     */
    let input = {
      "method": "approve",
      "params": {
        "operate": "apply",
        "item": "committee",
        "address":  "sourceAddress3"
      }
    }

    let contractInvoke = yield
sdk.operation.contractInvokeByGasOperation({
      contractAddress,
      sourceAddress: sourceAddress,
      gasAmount: '0',
      input: JSON.stringify(input),
    });
```



```
console.log(contractInvoke)

expect(contractInvoke.errorCode).to.equal(0)

const operationItem = contractInvoke.result.operation;

console.log(operationItem)

let feeData = yield sdk.transaction.evaluateFee({
  sourceAddress: sourceAddress,
  nonce,
  operations: [operationItem],
  signtureNumber: '100',
});
console.log(feeData)
expect(feeData.errorCode).to.equal(0)

let feeLimit = feeData.result.feeLimit;
let gasPrice = feeData.result.gasPrice;

console.log("gasPrice", gasPrice);
console.log("feeLimit", feeLimit);

const blobInfo = sdk.transaction.buildBlob({
  sourceAddress: sourceAddress,
  gasPrice: gasPrice,
  feeLimit: feeLimit,
  nonce: nonce,
  operations: [operationItem],
});

console.log(blobInfo);
expect(blobInfo.errorCode).to.equal(0)

const signed = sdk.transaction.sign({
  privateKeys: [privateKey],
  blob: blobInfo.result.transactionBlob
})

console.log(signed)
expect(signed.errorCode).to.equal(0)

let submitted = yield sdk.transaction.submit({
  signature: signed.result.signatures,
  blob: blobInfo.result.transactionBlob
})

console.log(submitted)
expect(submitted.errorCode).to.equal(0)

let info = null;
for (let i = 0; i < 10; i++) {
      console.log("Getting the transaction history (attempt " + (i +
1).toString() + ")...")
      info = yield sdk.transaction.getInfo(submitted.result.hash)
      if (info.errorCode === 0) {
        break;
      }
      sleep(2000);
    }

    expect(info.errorCode).to.equal(0)

  });

  it('query committee approve check', async () => {

    let data = await sdk.contract.call({
      optType: 2,
      contractAddress: contractAddress,
      input: JSON.stringify({
        method: 'getCommittee',
      }),
    });
    console.log(data)
    console.log(data.result.query_rets[0].result.value);
    expect(data.errorCode).to.equal(0);

expect(data.result.query_rets[0].result.value).to.equal('{"committee":["
sourceAddress","sourceAddress3"]}')
  });

  });
```

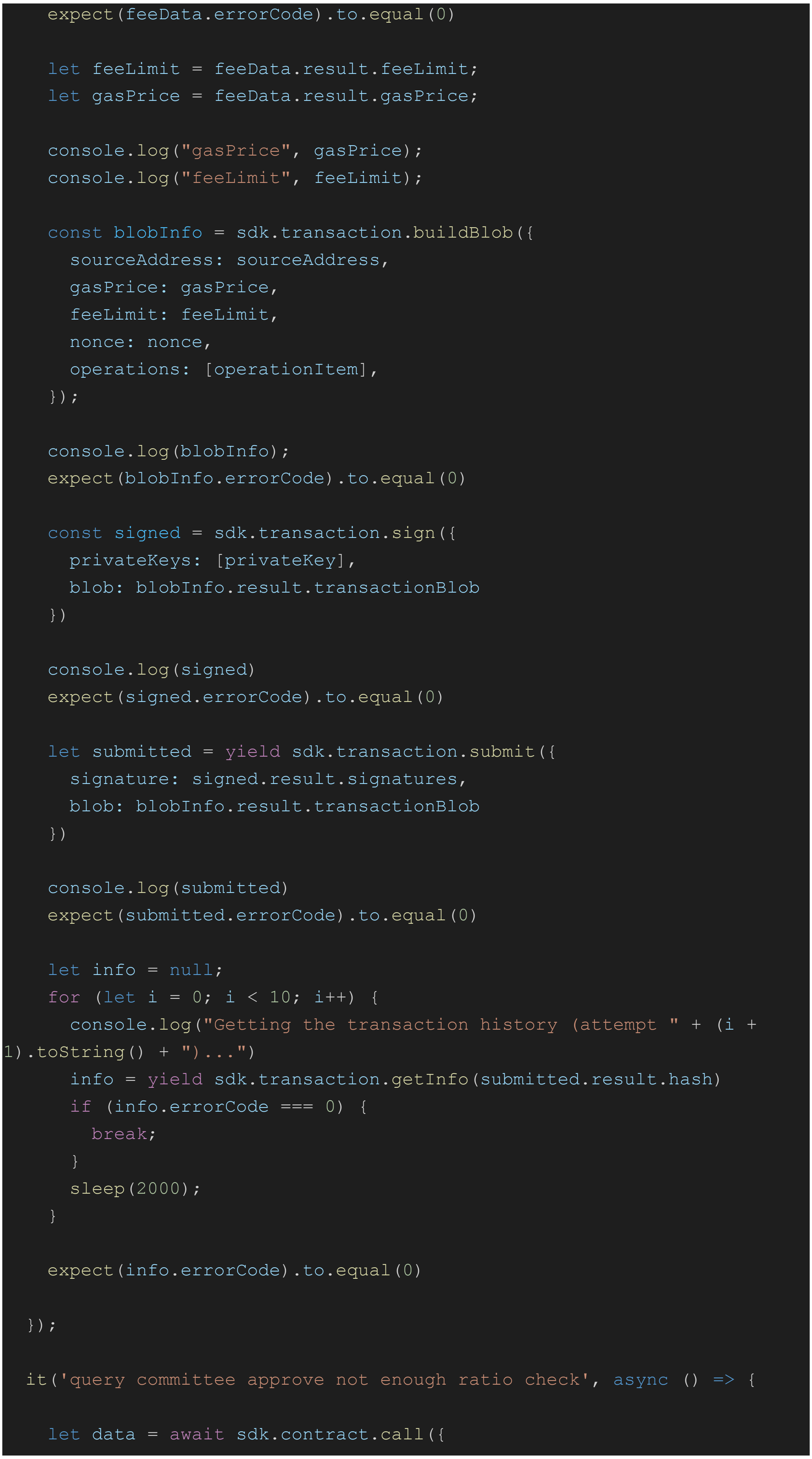
Expected Behavior	The getCommittee query raises the address of the newly approved committee member along with the existing member.  {"committee":["[redacted]","[redacted]"]} ✓ query committee approve check (1272ms)

4.5 Committee approve not enough ratio check

Result	**PASS**
Function	Apply and Approve

Objective	Verifies that the committee candidate is added into the committee only when the “pass_rate” is satisfied. Here, the “pass_rate” is not satisfied and thus expects an error.
Test Case	

```
// committee - sourceAddress, sourceAddress3; committee candidate -
sourceAddress2; pass rate - 0.66
describe('Apply and Approve test 5', async function () {
  this.timeout(30000);

  it('committee approve not enough ratio check', function* () {

    const nonceResult = yield sdk.account.getNonce(sourceAddress);

    expect(nonceResult.errorCode).to.equal(0)

    let nonce = nonceResult.result.nonce;
    nonce = new BigNumber(nonce).plus(1).toString(10);

    /*
     Specify the input parameters for invoking contract
     */
    let input = {
      "method": "approve",
      "params": {
        "operate": "apply",
        "item": "committee",
        "address":  "sourceAddress2"
      }
    }

    let contractInvoke = yield
sdk.operation.contractInvokeByGasOperation({
      contractAddress,
      sourceAddress: sourceAddress,
      gasAmount: '0',
      input: JSON.stringify(input),
    });

    console.log(contractInvoke)

    expect(contractInvoke.errorCode).to.equal(0)

    const operationItem = contractInvoke.result.operation;

    console.log(operationItem)

    let feeData = yield sdk.transaction.evaluateFee({
      sourceAddress: sourceAddress,
      nonce,
      operations: [operationItem],
      signtureNumber: '100',
    });
    console.log(feeData)
```

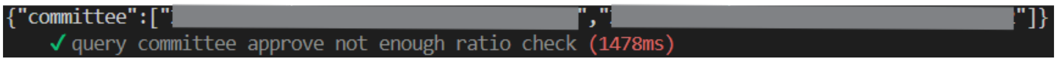


```
    expect(feeData.errorCode).to.equal(0)

    let feeLimit = feeData.result.feeLimit;
    let gasPrice = feeData.result.gasPrice;

    console.log("gasPrice", gasPrice);
    console.log("feeLimit", feeLimit);

    const blobInfo = sdk.transaction.buildBlob({
      sourceAddress: sourceAddress,
      gasPrice: gasPrice,
      feeLimit: feeLimit,
      nonce: nonce,
      operations: [operationItem],
    });

    console.log(blobInfo);
    expect(blobInfo.errorCode).to.equal(0)

    const signed = sdk.transaction.sign({
      privateKeys: [privateKey],
      blob: blobInfo.result.transactionBlob
    })

    console.log(signed)
    expect(signed.errorCode).to.equal(0)

    let submitted = yield sdk.transaction.submit({
      signature: signed.result.signatures,
      blob: blobInfo.result.transactionBlob
    })

    console.log(submitted)
    expect(submitted.errorCode).to.equal(0)

    let info = null;
    for (let i = 0; i < 10; i++) {
      console.log("Getting the transaction history (attempt " + (i +
1).toString() + ")...")
      info = yield sdk.transaction.getInfo(submitted.result.hash)
      if (info.errorCode === 0) {
        break;
      }
      sleep(2000);
    }

    expect(info.errorCode).to.equal(0)

  });

  it('query committee approve not enough ratio check', async () => {

    let data = await sdk.contract.call({
```

<table>
<tr><td></td><td>

```
   optType: 2,
   contractAddress: contractAddress,
   input: JSON.stringify({
    method: 'getCommittee',
   }),
  });
  console.log(data)
  console.log(data.result.query_rets[0].result.value);
  expect(data.errorCode).to.equal(0);

expect(data.result.query_rets[0].result.value).to.not.equal('{"committee
":["sourceAddress","sourceAddress3","sourceAddress2"]}')
 });

 });
```

</td></tr>
<tr><td><b>Expected Behavior</b></td><td>The committee applying candidate is not a member of this committee due to not enough ratio.



```
{"committee":["[redacted]","[redacted]"]}
   ✓ query committee approve not enough ratio check (1478ms)
```

</td></tr>
</table>

4.6 Committee approve enough ratio check

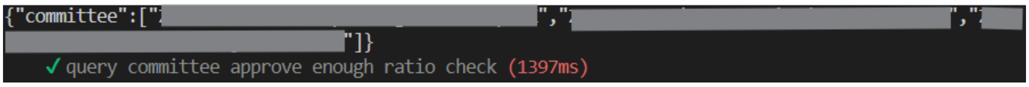
<table>
<tr><td><b>Result</b></td><td><b>PASS</b></td></tr>
<tr><td><b>Function</b></td><td>Apply and Approve</td></tr>
<tr><td><b>Objective</b></td><td>Checks the process of approving a committee role proposal when the “pass_rate” is satisfied. It approves the proposal with enough committee members to satisfy the pass_rate and verifies the result.</td></tr>
<tr><td><b>Test Case</b></td><td>

```
// committee - sourceAddress, sourceAddress3; committee candidate -
sourceAddress2; approved by - sourceAddress; pass rate - 0.66
describe('Apply and Approve test 6', async function () {
 this.timeout(30000);

 it('committee approve enough ratio check', function* () {

  const nonceResult = yield sdk.account.getNonce(sourceAddress3);

  expect(nonceResult.errorCode).to.equal(0)

  let nonce = nonceResult.result.nonce;
  nonce = new BigNumber(nonce).plus(1).toString(10);
```

</td></tr>
</table>

```
    /*
     Specify the input parameters for invoking contract
     */
    let input = {
      "method": "approve",
      "params": {
        "operate": "apply",
        "item": "committee",
        "address":  "sourceAddress2"
      }
    }

    let contractInvoke = yield
sdk.operation.contractInvokeByGasOperation({
      contractAddress,
      sourceAddress: sourceAddress3,
      gasAmount: '0',
      input: JSON.stringify(input),
    });

    console.log(contractInvoke)

    expect(contractInvoke.errorCode).to.equal(0)

    const operationItem = contractInvoke.result.operation;

    console.log(operationItem)

    let feeData = yield sdk.transaction.evaluateFee({
      sourceAddress: sourceAddress3,
      nonce,
      operations: [operationItem],
      signtureNumber: '100',
    });
    console.log(feeData)
    expect(feeData.errorCode).to.equal(0)

    let feeLimit = feeData.result.feeLimit;
    let gasPrice = feeData.result.gasPrice;

    console.log("gasPrice", gasPrice);
    console.log("feeLimit", feeLimit);

    const blobInfo = sdk.transaction.buildBlob({
      sourceAddress: sourceAddress3,
      gasPrice: gasPrice,
      feeLimit: feeLimit,
      nonce: nonce,
      operations: [operationItem],
    });

    console.log(blobInfo);
```

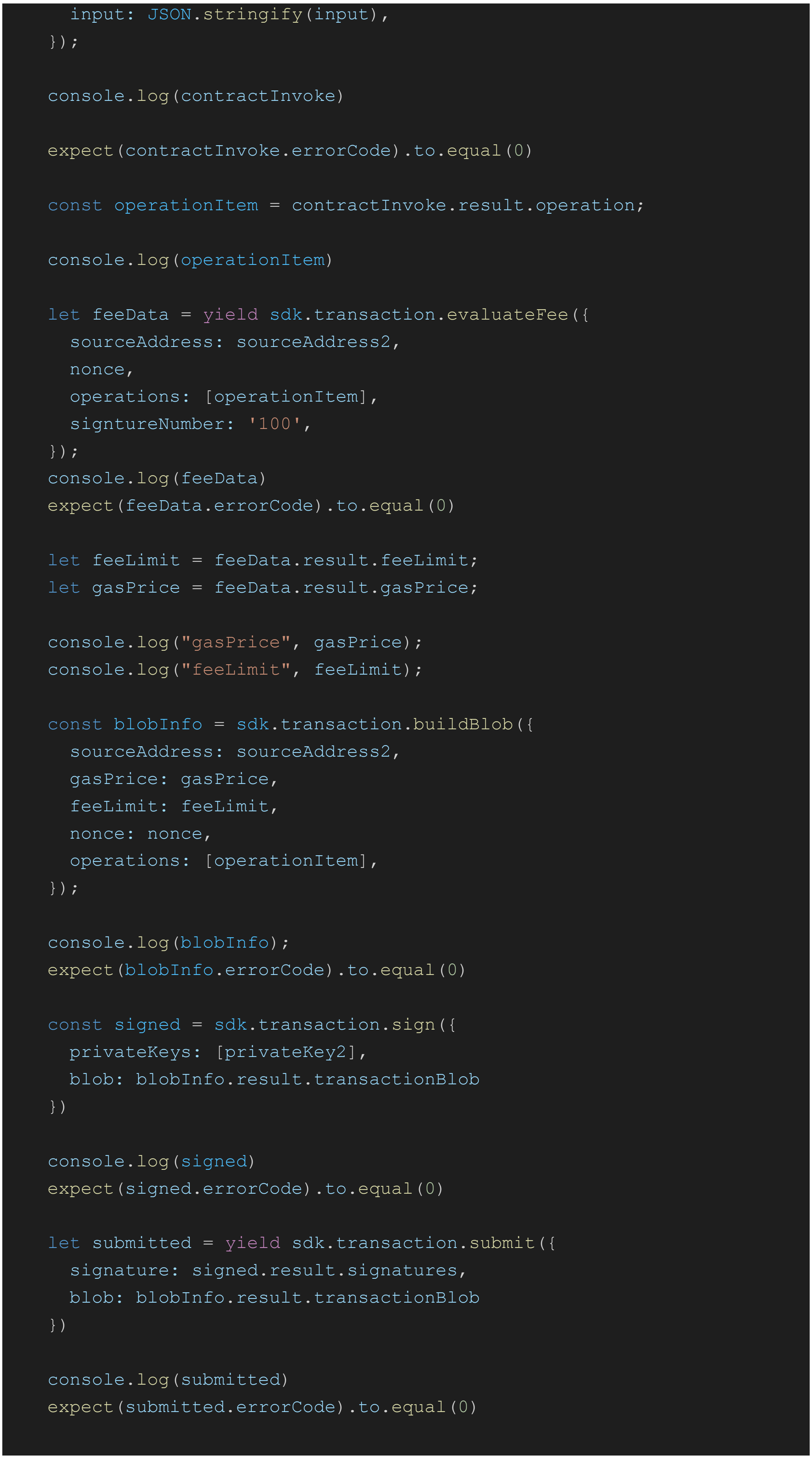
```
  expect(blobInfo.errorCode).to.equal(0)

  const signed = sdk.transaction.sign({
   privateKeys: [privateKey3],
   blob: blobInfo.result.transactionBlob
  })

  console.log(signed)
  expect(signed.errorCode).to.equal(0)

  let submitted = yield sdk.transaction.submit({
   signature: signed.result.signatures,
   blob: blobInfo.result.transactionBlob
  })

  console.log(submitted)
  expect(submitted.errorCode).to.equal(0)

  let info = null;
  for (let i = 0; i < 10; i++) {
   console.log("Getting the transaction history (attempt " + (i +
1).toString() + ")...")
   info = yield sdk.transaction.getInfo(submitted.result.hash)
   if (info.errorCode === 0) {
    break;
   }
   sleep(2000);
  }

  expect(info.errorCode).to.equal(0)

 });

 it('query committee approve enough ratio check', async () => {

  let data = await sdk.contract.call({
   optType: 2,
   contractAddress: contractAddress,
   input: JSON.stringify({
    method: 'getCommittee',
   }),
  });
  console.log(data)
  console.log(data.result.query_rets[0].result.value);
  expect(data.errorCode).to.equal(0);

expect(data.result.query_rets[0].result.value).to.equal('{"committee":["
sourceAddress","sourceAddress3","sourceAddress2"]}')
 });

 });
```

Expected Behavior	With enough ratio of committee members approving this proposal, the committee candidate is part of the committee.  {"committee":["[redacted]","[redacted]","[redacted] [redacted]"]} ✓ query committee approve enough ratio check (1397ms)

4.7 Committee abolish check

Result	**PASS**
Function	Abolish
Objective	Checks the process of abolishing a committee member. It invokes the "abolish" function with the specified parameters and verifies the result.
Test Case	

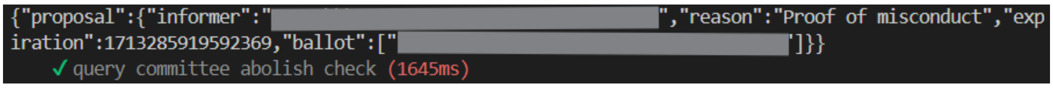
```
// committee - sourceAddress, sourceAddress2, sourceAddress3
describe('Abolish test 1', async function () {
  this.timeout(30000);

  it('committee abolish check', function* () {

    const nonceResult = yield sdk.account.getNonce(sourceAddress2);

    expect(nonceResult.errorCode).to.equal(0)

    let nonce = nonceResult.result.nonce;
    nonce = new BigNumber(nonce).plus(1).toString(10);

    /*
     Specify the input parameters for invoking contract
     */
    let input = {
      "method": "abolish",
      "params": {
        "role": "committee",
        "address": "sourceAddress3",
        "proof": "Proof of misconduct"
      }
    }


    let contractInvoke = yield
sdk.operation.contractInvokeByGasOperation({
      contractAddress,
      sourceAddress: sourceAddress2,
      gasAmount: '0',
```



```
  input: JSON.stringify(input),
});

console.log(contractInvoke)

expect(contractInvoke.errorCode).to.equal(0)

const operationItem = contractInvoke.result.operation;

console.log(operationItem)

let feeData = yield sdk.transaction.evaluateFee({
  sourceAddress: sourceAddress2,
  nonce,
  operations: [operationItem],
  signtureNumber: '100',
});
console.log(feeData)
expect(feeData.errorCode).to.equal(0)

let feeLimit = feeData.result.feeLimit;
let gasPrice = feeData.result.gasPrice;

console.log("gasPrice", gasPrice);
console.log("feeLimit", feeLimit);

const blobInfo = sdk.transaction.buildBlob({
  sourceAddress: sourceAddress2,
  gasPrice: gasPrice,
  feeLimit: feeLimit,
  nonce: nonce,
  operations: [operationItem],
});

console.log(blobInfo);
expect(blobInfo.errorCode).to.equal(0)

const signed = sdk.transaction.sign({
  privateKeys: [privateKey2],
  blob: blobInfo.result.transactionBlob
})

console.log(signed)
expect(signed.errorCode).to.equal(0)

let submitted = yield sdk.transaction.submit({
  signature: signed.result.signatures,
  blob: blobInfo.result.transactionBlob
})

console.log(submitted)
expect(submitted.errorCode).to.equal(0)
```

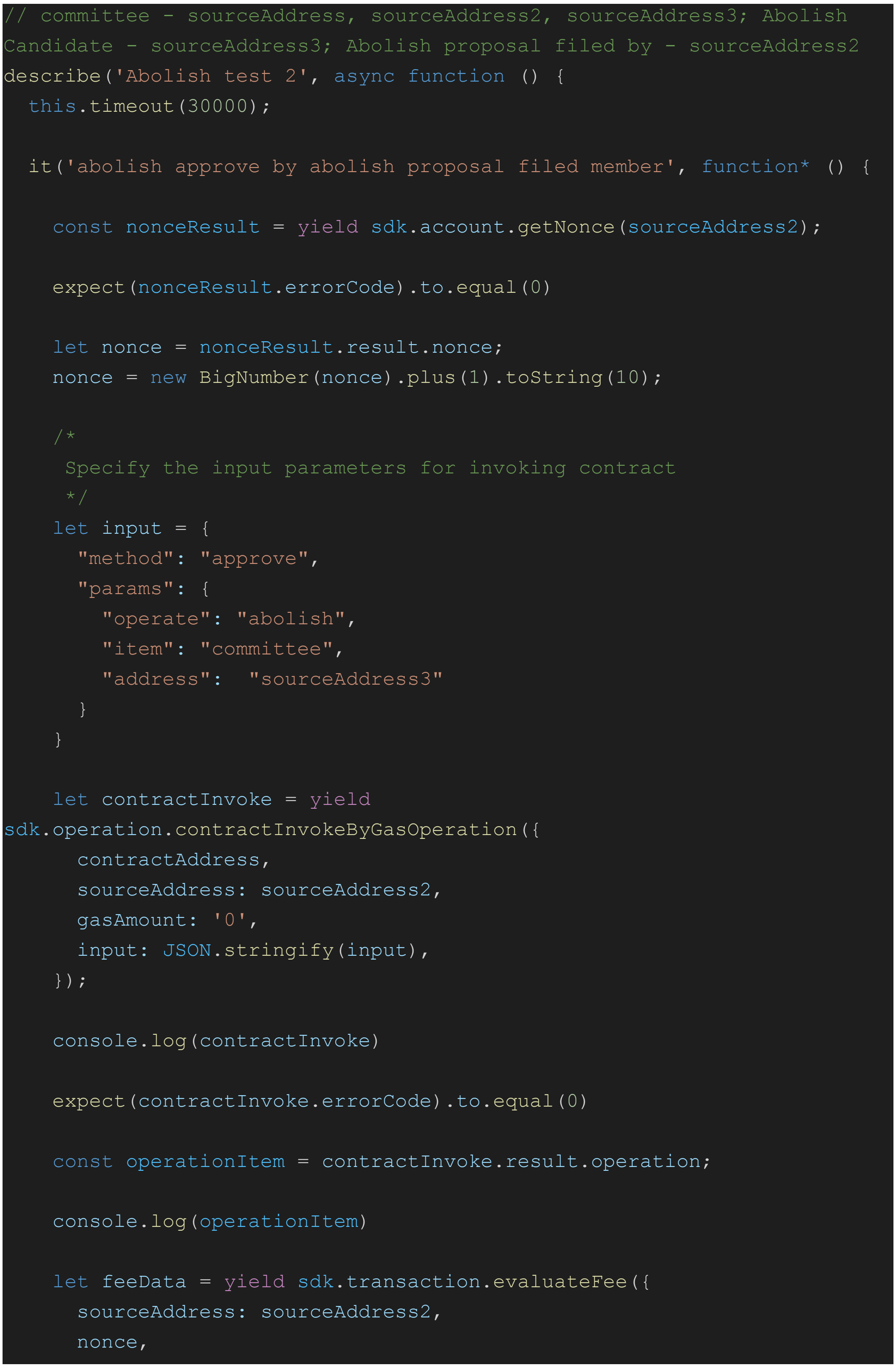
```
    let info = null;
    for (let i = 0; i < 10; i++) {
      console.log("Getting the transaction history (attempt " + (i +
1).toString() + ")...")
      info = yield sdk.transaction.getInfo(submitted.result.hash)
      if (info.errorCode === 0) {
        break;
      }
      sleep(2000);
    }

    expect(info.errorCode).to.equal(0)

  });

  it('query committee abolish check', async () => {

    let data = await sdk.contract.call({
      optType: 2,
      contractAddress: contractAddress,
      input: JSON.stringify({
        method: 'getProposal',
           params: {
             operate: 'abolish',
             item: 'committee',
             address: 'sourceAddress3'
           }
      }),
    });
    console.log(data)
    console.log(data.result.query_rets[0].result.value);
    expect(data.errorCode).to.equal(0);

expect(data.result.query_rets[0].result.value).to.not.equal('{"proposal"
:false}');
  });

  });
```

Expected Behavior	As "proposal" is not false, the user has successfully applied to abolish a fellow committee member.



```
{"proposal":{"informer":"[redacted]","reason":"Proof of misconduct","exp
iration":1713285919592369,"ballot":["[redacted]"]}}
    ✓ query committee abolish check (1645ms)
```

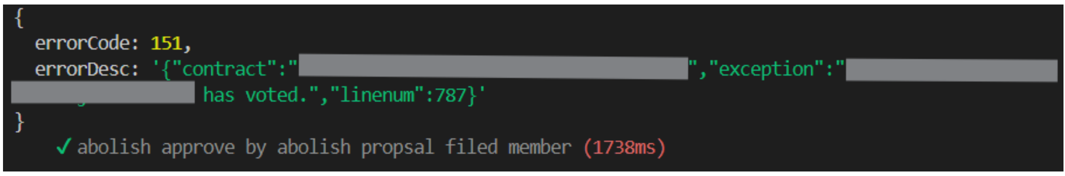
4.8 Abolish approve by abolish proposal filed member

Result	PASS
Function	Abolish
Objective	Verifies that “abolish” can’t be approved by the “abolish” proposed member. Here, the member who made the abolish proposal tries to approve it and thus expects an error.
Test Case	

```
// committee - sourceAddress, sourceAddress2, sourceAddress3; Abolish
Candidate - sourceAddress3; Abolish proposal filed by - sourceAddress2
describe('Abolish test 2', async function () {
  this.timeout(30000);

  it('abolish approve by abolish proposal filed member', function* () {

    const nonceResult = yield sdk.account.getNonce(sourceAddress2);

    expect(nonceResult.errorCode).to.equal(0)

    let nonce = nonceResult.result.nonce;
    nonce = new BigNumber(nonce).plus(1).toString(10);

    /*
     Specify the input parameters for invoking contract
     */
    let input = {
      "method": "approve",
      "params": {
        "operate": "abolish",
        "item": "committee",
        "address":  "sourceAddress3"
      }
    }

    let contractInvoke = yield
sdk.operation.contractInvokeByGasOperation({
      contractAddress,
      sourceAddress: sourceAddress2,
      gasAmount: '0',
      input: JSON.stringify(input),
    });

    console.log(contractInvoke)

    expect(contractInvoke.errorCode).to.equal(0)

    const operationItem = contractInvoke.result.operation;

    console.log(operationItem)

    let feeData = yield sdk.transaction.evaluateFee({
      sourceAddress: sourceAddress2,
      nonce,
      operations: [operationItem],
      signtureNumber: '100',
    });
    console.log(feeData)
    expect(feeData.errorCode).to.equal(151)


  });


  });
```

Expected Behavior

The same committee member who raised the abolish proposal, trying to approve it, raises an error.



```
{
  errorCode: 151,
  errorDesc: '{"contract":"[redacted]","exception":"[redacted]
[redacted] has voted.","linenum":787}'
}
    ✓ abolish approve by abolish propsal filed member (1738ms)
```

4.9 Abolish approve check

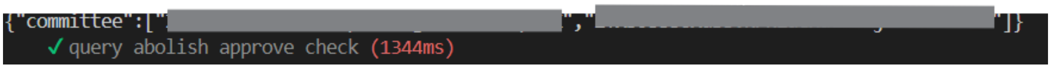
Result	**PASS**
Function	Abolish
Objective	Checks the process of approving the abolish proposal of a committee member. It invokes the "approve" function with the specified parameters and verifies the result.

Test Case

```
// committee - sourceAddress, sourceAddress2, sourceAddress3; Abolish
Candidate - sourceAddress3; Abolish proposal filed by - sourceAddress2
describe('Abolish test 3', async function () {
  this.timeout(30000);

  it('abolish approve check', function* () {

    const nonceResult = yield sdk.account.getNonce(sourceAddress);

    expect(nonceResult.errorCode).to.equal(0)

    let nonce = nonceResult.result.nonce;
    nonce = new BigNumber(nonce).plus(1).toString(10);

    /*
     Specify the input parameters for invoking contract
     */
    let input = {
      "method": "approve",
      "params": {
        "operate": "abolish",
        "item": "committee",
        "address":  "sourceAddress3"
      }
    }

    let contractInvoke = yield
sdk.operation.contractInvokeByGasOperation({
      contractAddress,
      sourceAddress: sourceAddress,
      gasAmount: '0',
      input: JSON.stringify(input),
    });

    console.log(contractInvoke)

    expect(contractInvoke.errorCode).to.equal(0)

    const operationItem = contractInvoke.result.operation;

    console.log(operationItem)

    let feeData = yield sdk.transaction.evaluateFee({
      sourceAddress: sourceAddress,
      nonce,
      operations: [operationItem],
      signtureNumber: '100',
    });
    console.log(feeData)
    expect(feeData.errorCode).to.equal(0)

    let feeLimit = feeData.result.feeLimit;
    let gasPrice = feeData.result.gasPrice;

    console.log("gasPrice", gasPrice);
    console.log("feeLimit", feeLimit);

    const blobInfo = sdk.transaction.buildBlob({
      sourceAddress: sourceAddress,
      gasPrice: gasPrice,
      feeLimit: feeLimit,
      nonce: nonce,
      operations: [operationItem],
    });

    console.log(blobInfo);
    expect(blobInfo.errorCode).to.equal(0)

    const signed = sdk.transaction.sign({
```

```
   privateKeys: [privateKey],
   blob: blobInfo.result.transactionBlob
  })

  console.log(signed)
  expect(signed.errorCode).to.equal(0)

  let submitted = yield sdk.transaction.submit({
   signature: signed.result.signatures,
   blob: blobInfo.result.transactionBlob
  })

  console.log(submitted)
  expect(submitted.errorCode).to.equal(0)

  let info = null;
  for (let i = 0; i < 10; i++) {
   console.log("Getting the transaction history (attempt " + (i +
1).toString() + ")...")
   info = yield sdk.transaction.getInfo(submitted.result.hash)
   if (info.errorCode === 0) {
    break;
   }
   sleep(2000);
  }

  expect(info.errorCode).to.equal(0)

 });

 it('query abolish approve check', async () => {

  let data = await sdk.contract.call({
   optType: 2,
   contractAddress: contractAddress,
   input: JSON.stringify({
    method: 'getCommittee',
   }),
  });
  console.log(data)
  console.log(data.result.query_rets[0].result.value);
  expect(data.errorCode).to.equal(0);

expect(data.result.query_rets[0].result.value).to.equal('{"committee":["
sourceAddress","sourceAddress2"]}')
 });



 });
```

<table>
<tr><td><b>Expected Behavior</b></td><td>The abolished committee member is not supposed to appear when the getCommittee query is called.

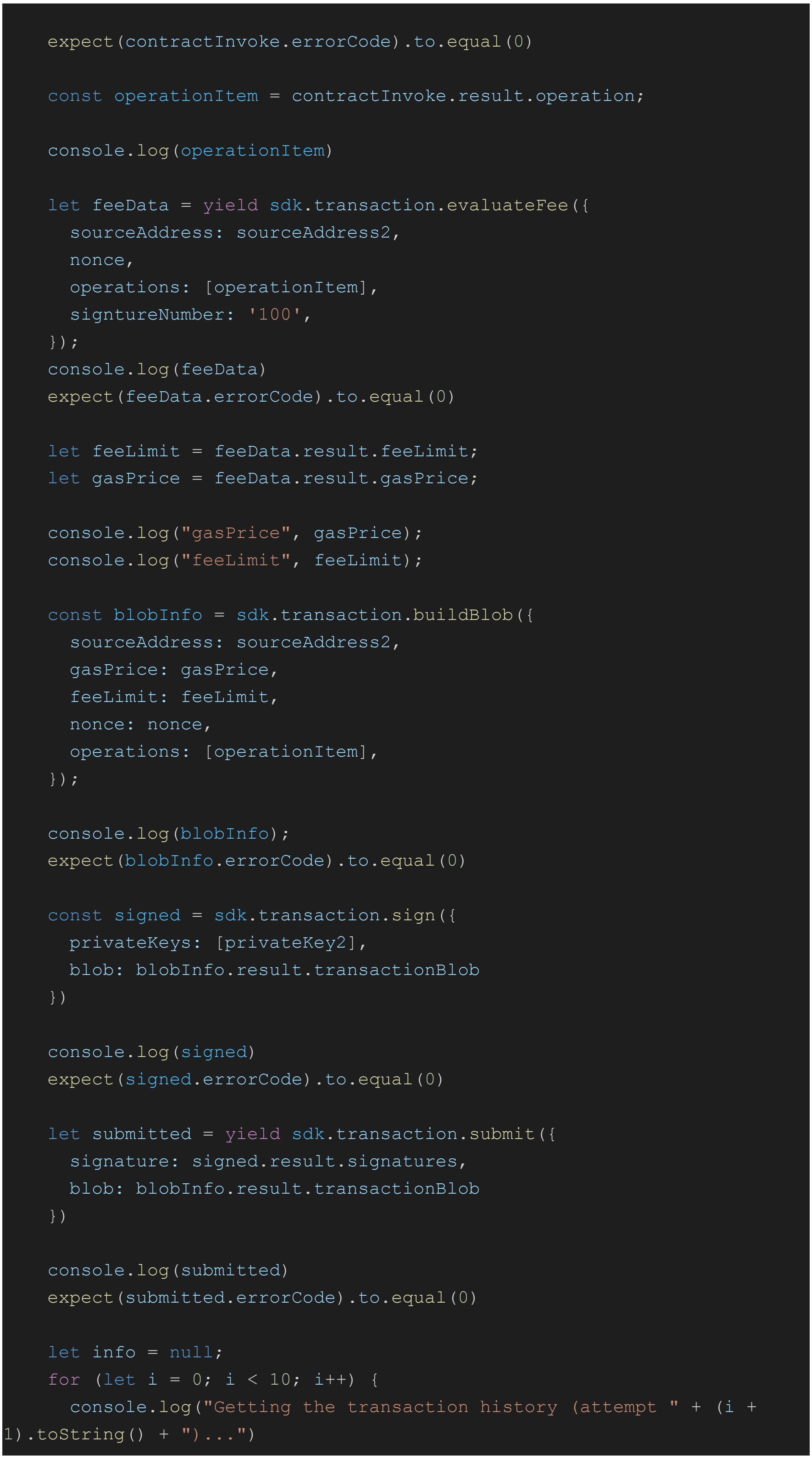


```
{"committee":["[redacted]","[redacted]"]}
    ✓ query abolish approve check (1344ms)
```
</td></tr>
</table>

4.10 Configure proposal check

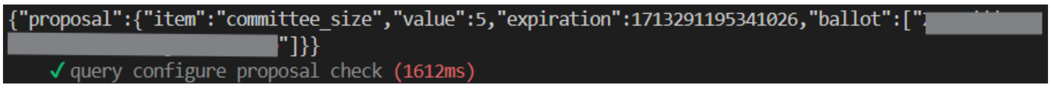
<table>
<tr><td><b>Result</b></td><td><b>PASS</b></td></tr>
<tr><td><b>Function</b></td><td>Configure</td></tr>
<tr><td><b>Objective</b></td><td>Checks the process of changing the configuration of the smart contract. It invokes the "configure" function to make a proposal and verifies the result.</td></tr>
<tr><td><b>Test Case</b></td><td>

```
//committee - sourceAddress, sourceAddress2
describe('Configure test 1', async function () {
  this.timeout(30000);

  it('configure proposal check', function* () {

    const nonceResult = yield sdk.account.getNonce(sourceAddress2);

    expect(nonceResult.errorCode).to.equal(0)

    let nonce = nonceResult.result.nonce;
    nonce = new BigNumber(nonce).plus(1).toString(10);

    /*
     Specify the input parameters for invoking contract
     */
    let input = {
      "method": "configure",
      "params": {
        "item": "committee_size",
        "value": 5
      }

    }

    let contractInvoke = yield
sdk.operation.contractInvokeByGasOperation({
      contractAddress,
      sourceAddress: sourceAddress2,
      gasAmount: '0',
      input: JSON.stringify(input),
    });

    console.log(contractInvoke)
```
</td></tr>
</table>



```
    expect(contractInvoke.errorCode).to.equal(0)

    const operationItem = contractInvoke.result.operation;

    console.log(operationItem)

    let feeData = yield sdk.transaction.evaluateFee({
      sourceAddress: sourceAddress2,
      nonce,
      operations: [operationItem],
      signtureNumber: '100',
    });
    console.log(feeData)
    expect(feeData.errorCode).to.equal(0)

    let feeLimit = feeData.result.feeLimit;
    let gasPrice = feeData.result.gasPrice;

    console.log("gasPrice", gasPrice);
    console.log("feeLimit", feeLimit);

    const blobInfo = sdk.transaction.buildBlob({
      sourceAddress: sourceAddress2,
      gasPrice: gasPrice,
      feeLimit: feeLimit,
      nonce: nonce,
      operations: [operationItem],
    });

    console.log(blobInfo);
    expect(blobInfo.errorCode).to.equal(0)

    const signed = sdk.transaction.sign({
      privateKeys: [privateKey2],
      blob: blobInfo.result.transactionBlob
    })

    console.log(signed)
    expect(signed.errorCode).to.equal(0)

    let submitted = yield sdk.transaction.submit({
      signature: signed.result.signatures,
      blob: blobInfo.result.transactionBlob
    })

    console.log(submitted)
    expect(submitted.errorCode).to.equal(0)

    let info = null;
    for (let i = 0; i < 10; i++) {
      console.log("Getting the transaction history (attempt " + (i +
1).toString() + ")...")
```

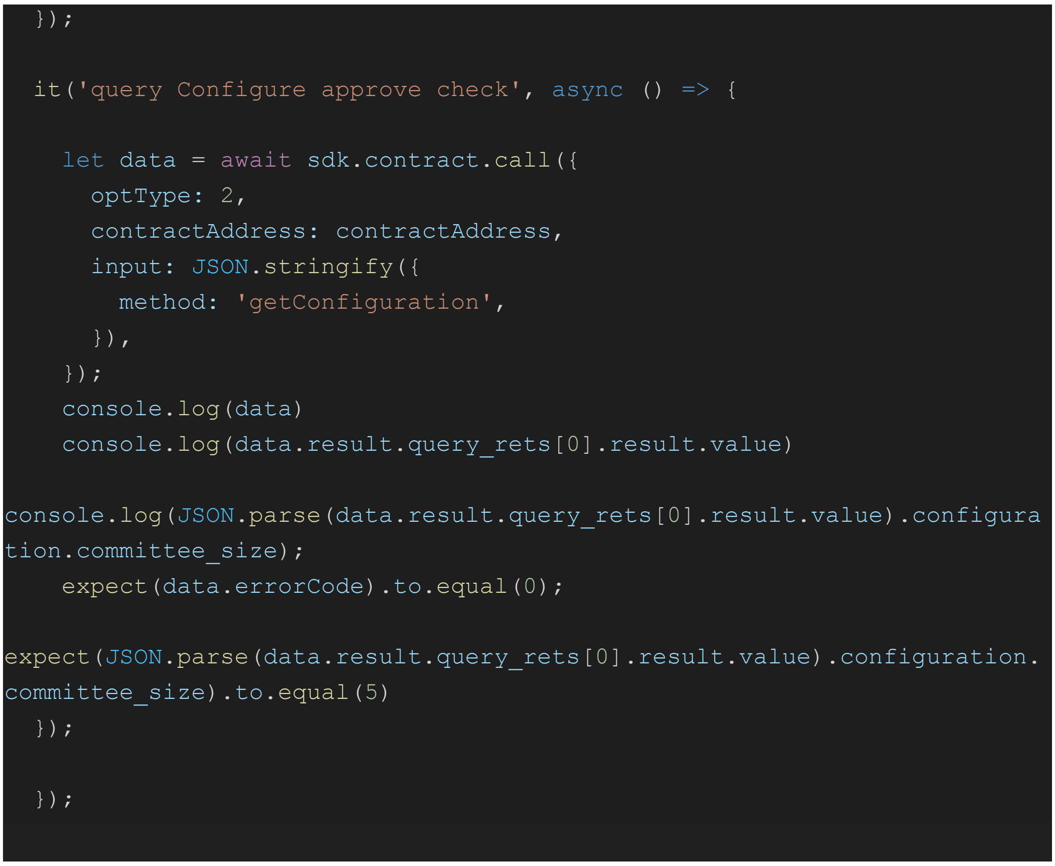
```
      info = yield sdk.transaction.getInfo(submitted.result.hash)
      if (info.errorCode === 0) {
        break;
      }
      sleep(2000);
    }

    expect(info.errorCode).to.equal(0)

  });

  it('query configure proposal check', async () => {

    let data = await sdk.contract.call({
      optType: 2,
      contractAddress: contractAddress,
      input: JSON.stringify({
        method: 'getProposal',
           params: {
             operate: 'config',
             item: 'committee_size',
             address: 'sourceAddress2'
           }
      }),
    });
    console.log(data)
    console.log(data.result.query_rets[0].result.value);
    expect(data.errorCode).to.equal(0);

expect(data.result.query_rets[0].result.value).to.not.equal('{"proposal"
:false}');
  });


  });
```

Expected Behavior	As "proposal" is not false, the user has successfully applied to change the configuration of the contract.  {"proposal":{"item":"committee_size","value":5,"expiration":1713291195341026,"ballot":["[redacted]"]}} ✓ query configure proposal check (1612ms)

4.11 Configure approve check

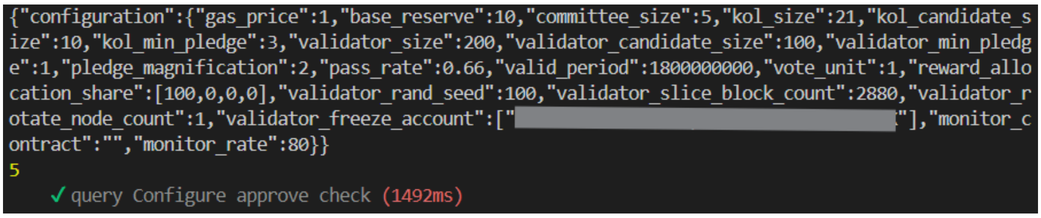
Result	**PASS**
Function	Configure
Objective	Checks the process of changing the configuration of the smart contract. It invokes the "approve" function to approve the config proposal and verifies the result.
Test Case	

```
//committee - sourceAddress, sourceAddress2; Configure proposal -
sourceAddress2
describe('Configure test 2', async function () {
  this.timeout(30000);

  it('Configure approve check', function* () {

    const nonceResult = yield sdk.account.getNonce(sourceAddress);

    expect(nonceResult.errorCode).to.equal(0)

    let nonce = nonceResult.result.nonce;
    nonce = new BigNumber(nonce).plus(1).toString(10);

    /*
     Specify the input parameters for invoking contract
     */
    let input = {
      "method": "approve",
      "params": {
        "operate": "config",
        "item": "committee_size",
        "address":  "sourceAddress2"
      }
    }


    let contractInvoke = yield
sdk.operation.contractInvokeByGasOperation({
      contractAddress,
      sourceAddress: sourceAddress,
      gasAmount: '0',
      input: JSON.stringify(input),
    });

    console.log(contractInvoke)

    expect(contractInvoke.errorCode).to.equal(0)

    const operationItem = contractInvoke.result.operation;

    console.log(operationItem)

    let feeData = yield sdk.transaction.evaluateFee({
      sourceAddress: sourceAddress,
      nonce,
      operations: [operationItem],
      signtureNumber: '100',
    });
    console.log(feeData)
    expect(feeData.errorCode).to.equal(0)

    let feeLimit = feeData.result.feeLimit;
    let gasPrice = feeData.result.gasPrice;

    console.log("gasPrice", gasPrice);
    console.log("feeLimit", feeLimit);

    const blobInfo = sdk.transaction.buildBlob({
      sourceAddress: sourceAddress,
      gasPrice: gasPrice,
      feeLimit: feeLimit,
      nonce: nonce,
      operations: [operationItem],
    });

    console.log(blobInfo);
    expect(blobInfo.errorCode).to.equal(0)

    const signed = sdk.transaction.sign({
      privateKeys: [privateKey],
      blob: blobInfo.result.transactionBlob
    })

    console.log(signed)
    expect(signed.errorCode).to.equal(0)

    let submitted = yield sdk.transaction.submit({
      signature: signed.result.signatures,
      blob: blobInfo.result.transactionBlob
    })

    console.log(submitted)
    expect(submitted.errorCode).to.equal(0)

    let info = null;
    for (let i = 0; i < 10; i++) {
      console.log("Getting the transaction history (attempt " + (i +
1).toString() + ")...")
      info = yield sdk.transaction.getInfo(submitted.result.hash)
      if (info.errorCode === 0) {
        break;
      }
      sleep(2000);
    }

    expect(info.errorCode).to.equal(0)
```



```
  });

  it('query Configure approve check', async () => {

    let data = await sdk.contract.call({
      optType: 2,
      contractAddress: contractAddress,
      input: JSON.stringify({
        method: 'getConfiguration',
      }),
    });
    console.log(data)
    console.log(data.result.query_rets[0].result.value)

console.log(JSON.parse(data.result.query_rets[0].result.value).configura
tion.committee_size);
    expect(data.errorCode).to.equal(0);

expect(JSON.parse(data.result.query_rets[0].result.value).configuration.
committee_size).to.equal(5)
  });

  });
```

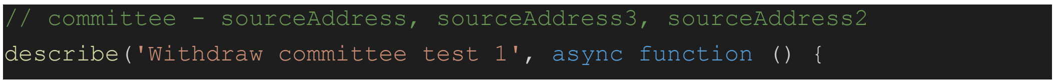
Expected Behavior	As the user's configure proposal to change the committee_size is accepted the getConfiguration query raises the current committee_size as 5.



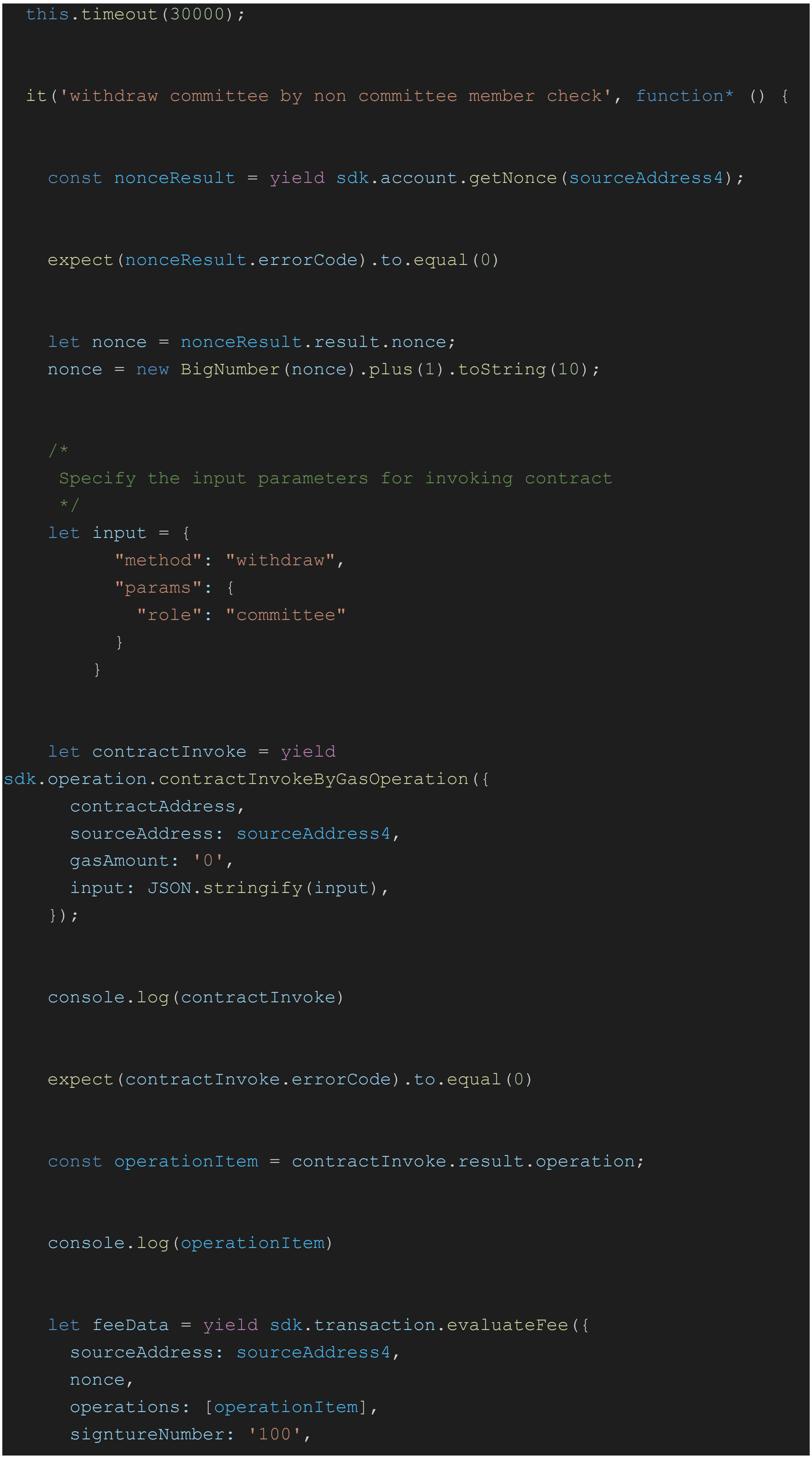
```
{"configuration":{"gas_price":1,"base_reserve":10,"committee_size":5,"kol_size":21,"kol_candidate_s
ize":10,"kol_min_pledge":3,"validator_size":200,"validator_candidate_size":100,"validator_min_pledg
e":1,"pledge_magnification":2,"pass_rate":0.66,"valid_period":1800000000,"vote_unit":1,"reward_allo
cation_share":[100,0,0,0],"validator_rand_seed":100,"validator_slice_block_count":2880,"validator_r
otate_node_count":1,"validator_freeze_account":["[redacted]"],"monitor_c
ontract":"","monitor_rate":80}}
5
    ✓ query Configure approve check (1492ms)
```

4.12 Withdraw committee by non committee member check

Result	**PASS**
Function	Withdraw
Objective	Verifies that the "Withdraw" function can be only called by existing members of the contract and cannot be called by non committee member.
Test Case	



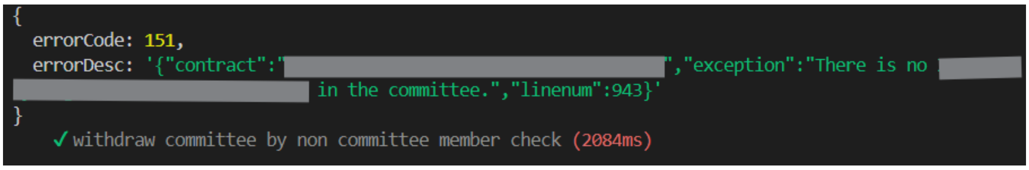
```
// committee - sourceAddress, sourceAddress3, sourceAddress2
describe('Withdraw committee test 1', async function () {
```



```
  this.timeout(30000);


  it('withdraw committee by non committee member check', function* () {


    const nonceResult = yield sdk.account.getNonce(sourceAddress4);


    expect(nonceResult.errorCode).to.equal(0)


    let nonce = nonceResult.result.nonce;
    nonce = new BigNumber(nonce).plus(1).toString(10);


    /*
     Specify the input parameters for invoking contract
     */
    let input = {
          "method": "withdraw",
          "params": {
            "role": "committee"
          }
        }


    let contractInvoke = yield
sdk.operation.contractInvokeByGasOperation({
      contractAddress,
      sourceAddress: sourceAddress4,
      gasAmount: '0',
      input: JSON.stringify(input),
    });


    console.log(contractInvoke)


    expect(contractInvoke.errorCode).to.equal(0)


    const operationItem = contractInvoke.result.operation;


    console.log(operationItem)


    let feeData = yield sdk.transaction.evaluateFee({
      sourceAddress: sourceAddress4,
      nonce,
      operations: [operationItem],
      signtureNumber: '100',
```

<table>
<tr><td></td><td><pre>
    });
    console.log(feeData)
    expect(feeData.errorCode).to.equal(151)

  });

  });
</pre></td></tr>
<tr><td><b>Expected Behavior</b></td><td>A non-committee member trying to withdraw from the committee role raises an error.
<pre>
{
  errorCode: 151,
  errorDesc: '{"contract":"[redacted]","exception":"There is no [redacted]
[redacted] in the committee.","linenum":943}'
}
    ✓ withdraw committee by non committee member check (2084ms)
</pre></td></tr>
</table>

4.13 Withdraw committee wrong role

<table>
<tr><td><b>Result</b></td><td><b>PASS</b></td></tr>
<tr><td><b>Function</b></td><td>Withdraw</td></tr>
<tr><td><b>Objective</b></td><td>Verifies that the “Withdraw” function fails when a committee member tries to withdraw from a validator role.</td></tr>
<tr><td><b>Test Case</b></td><td><pre>
// committee - sourceAddress, sourceAddress3, sourceAddress2
describe('Withdraw committee test 2', async function () {
  this.timeout(30000);

  it('withdraw committee wrong role', function* () {

    const nonceResult = yield sdk.account.getNonce(sourceAddress2);

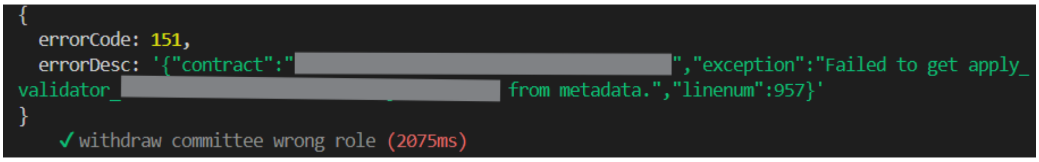
    expect(nonceResult.errorCode).to.equal(0)

    let nonce = nonceResult.result.nonce;
    nonce = new BigNumber(nonce).plus(1).toString(10);

    /*
     Specify the input parameters for invoking contract
</pre></td></tr>
</table>

```
     */
    let input = {
          "method": "withdraw",
          "params": {
            "role": "validator"
          }
        }


    let contractInvoke = yield
sdk.operation.contractInvokeByGasOperation({
      contractAddress,
      sourceAddress: sourceAddress2,
      gasAmount: '0',
      input: JSON.stringify(input),
    });


    console.log(contractInvoke)


    expect(contractInvoke.errorCode).to.equal(0)


    const operationItem = contractInvoke.result.operation;


    console.log(operationItem)


    let feeData = yield sdk.transaction.evaluateFee({
      sourceAddress: sourceAddress2,
      nonce,
      operations: [operationItem],
      signtureNumber: '100',
    });
    console.log(feeData)
    expect(feeData.errorCode).to.equal(151)

  });


  });
```

Expected Behavior	A committee member who is not a validator trying to withdraw from a validator role raises an error.



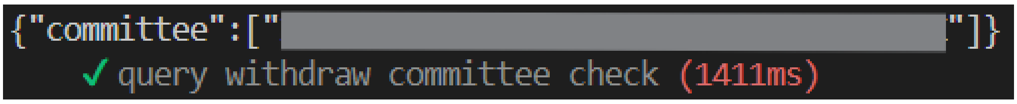
```
{
  errorCode: 151,
  errorDesc: '{"contract":"[redacted]","exception":"Failed to get apply_
validator_[redacted] from metadata.","linenum":957}'
}
    ✓ withdraw committee wrong role (2075ms)
```

4.14Withdraw committee check

Result	**PASS**
Function	Withdraw
Objective	Checks the process of withdrawing a committee member from the smart contract. It invokes the "withdraw” function and verifies the result.
Test Case	

```
// committee - sourceAddress, sourceAddress3, sourceAddress2
describe('Withdraw committee test 3', async function () {
  this.timeout(30000);

  it('withdraw committee check', function* () {

    const nonceResult = yield sdk.account.getNonce(sourceAddress2);

    expect(nonceResult.errorCode).to.equal(0)

    let nonce = nonceResult.result.nonce;
    nonce = new BigNumber(nonce).plus(1).toString(10);

    /*
     Specify the input parameters for invoking contract
     */
    let input = {
          "method": "withdraw",
          "params": {
            "role": "committee"
          }
        }

    let contractInvoke = yield
sdk.operation.contractInvokeByGasOperation({
      contractAddress,
  sourceAddress: sourceAddress2,
  gasAmount: '0',
  input: JSON.stringify(input),
});

console.log(contractInvoke)

expect(contractInvoke.errorCode).to.equal(0)

const operationItem = contractInvoke.result.operation;

console.log(operationItem)

let feeData = yield sdk.transaction.evaluateFee({
  sourceAddress: sourceAddress2,
  nonce,
  operations: [operationItem],
  signtureNumber: '100',
});
console.log(feeData)
expect(feeData.errorCode).to.equal(0)

let feeLimit = feeData.result.feeLimit;
let gasPrice = feeData.result.gasPrice;

console.log("gasPrice", gasPrice);
console.log("feeLimit", feeLimit);

const blobInfo = sdk.transaction.buildBlob({
  sourceAddress: sourceAddress2,
  gasPrice: gasPrice,
  feeLimit: feeLimit,
  nonce: nonce,
  operations: [operationItem],
});

console.log(blobInfo);
expect(blobInfo.errorCode).to.equal(0)

const signed = sdk.transaction.sign({
  privateKeys: [privateKey2],
  blob: blobInfo.result.transactionBlob
})

console.log(signed)
expect(signed.errorCode).to.equal(0)

let submitted = yield sdk.transaction.submit({
  signature: signed.result.signatures,
  blob: blobInfo.result.transactionBlob
})

console.log(submitted)
```

```
    expect(submitted.errorCode).to.equal(0)

    let info = null;
    for (let i = 0; i < 10; i++) {
      console.log("Getting the transaction history (attempt " + (i +
1).toString() + ")...")
      info = yield sdk.transaction.getInfo(submitted.result.hash)
      if (info.errorCode === 0) {
        break;
      }
      sleep(2000);
    }

    expect(info.errorCode).to.equal(0)

  });

  it('query withdraw committee check', async () => {

    let data = await sdk.contract.call({
      optType: 2,
      contractAddress: contractAddress,
      input: JSON.stringify({
        method: 'getCommittee',
      }),
    });
    console.log(data)
    console.log(data.result.query_rets[0].result.value);
    expect(data.errorCode).to.equal(0);

expect(data.result.query_rets[0].result.value).to.equal('{"committee":["
sourceAddress"]}')
  });

  });
```

Expected Behavior	A committee member withdraws and is no longer visible in the getCommittee query.  {"committee":["[redacted]"]} ✓ query withdraw committee check (1411ms)

4.15 Committee apply amount == 0 check

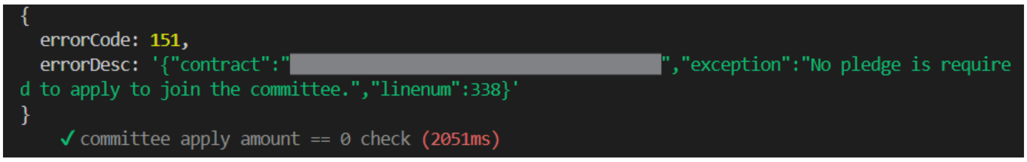
Result	**PASS**
Function	Apply and Approve

Objective	Verifies that the apply committee functions raises an error if the zetrix amount is not equal to 0.

Test Case

```
// committee - sourceAddress
describe('Apply and Approve test 7', async function () {
  this.timeout(30000);

  it('committee apply amount == 0 check', function* () {

    const nonceResult = yield sdk.account.getNonce(sourceAddress4);

    expect(nonceResult.errorCode).to.equal(0)

    let nonce = nonceResult.result.nonce;
    nonce = new BigNumber(nonce).plus(1).toString(10);

    /*
     Specify the input parameters for invoking contract
     */
    let input = {
      "method": "apply",
      "params": {
        "role": "committee",
        "ratio": 0,
        "node": "sourceAddress4"
      }
    }

    let contractInvoke = yield
sdk.operation.contractInvokeByGasOperation({
      contractAddress,
      sourceAddress: sourceAddress4,
      gasAmount: '1',
      input: JSON.stringify(input),
    });

    console.log(contractInvoke)

    expect(contractInvoke.errorCode).to.equal(0)

    const operationItem = contractInvoke.result.operation;

    console.log(operationItem)

    let feeData = yield sdk.transaction.evaluateFee({
      sourceAddress: sourceAddress4,
      nonce,
      operations: [operationItem],
      signtureNumber: '100',
    });
    console.log(feeData)
    expect(feeData.errorCode).to.equal(151)
```

<table>
<tr><td></td><td><pre>
  });

  });
</pre></td></tr>
<tr><td><b>Expected Behavior</b></td><td>The gas amount not being equal to 0 to apply for a committee role raises an error.
<pre>
{
  errorCode: 151,
  errorDesc: '{"contract":"[redacted]","exception":"No pledge is require
d to apply to join the committee.","linenum":338}'
}
    ✓ committee apply amount == 0 check (2051ms)
</pre></td></tr>
</table>

4.16 Validator apply amount != 0 check

<table>
<tr><td><b>Result</b></td><td><b>PASS</b></td></tr>
<tr><td><b>Function</b></td><td>Apply and Approve</td></tr>
<tr><td><b>Objective</b></td><td>Verifies that the apply validator functions raises an error if the zetrix amount is equal to 0.</td></tr>
<tr><td><b>Test Case</b></td><td><pre>
// committee - sourceAddress
describe('Apply and Approve test 8', async function () {
  this.timeout(30000);

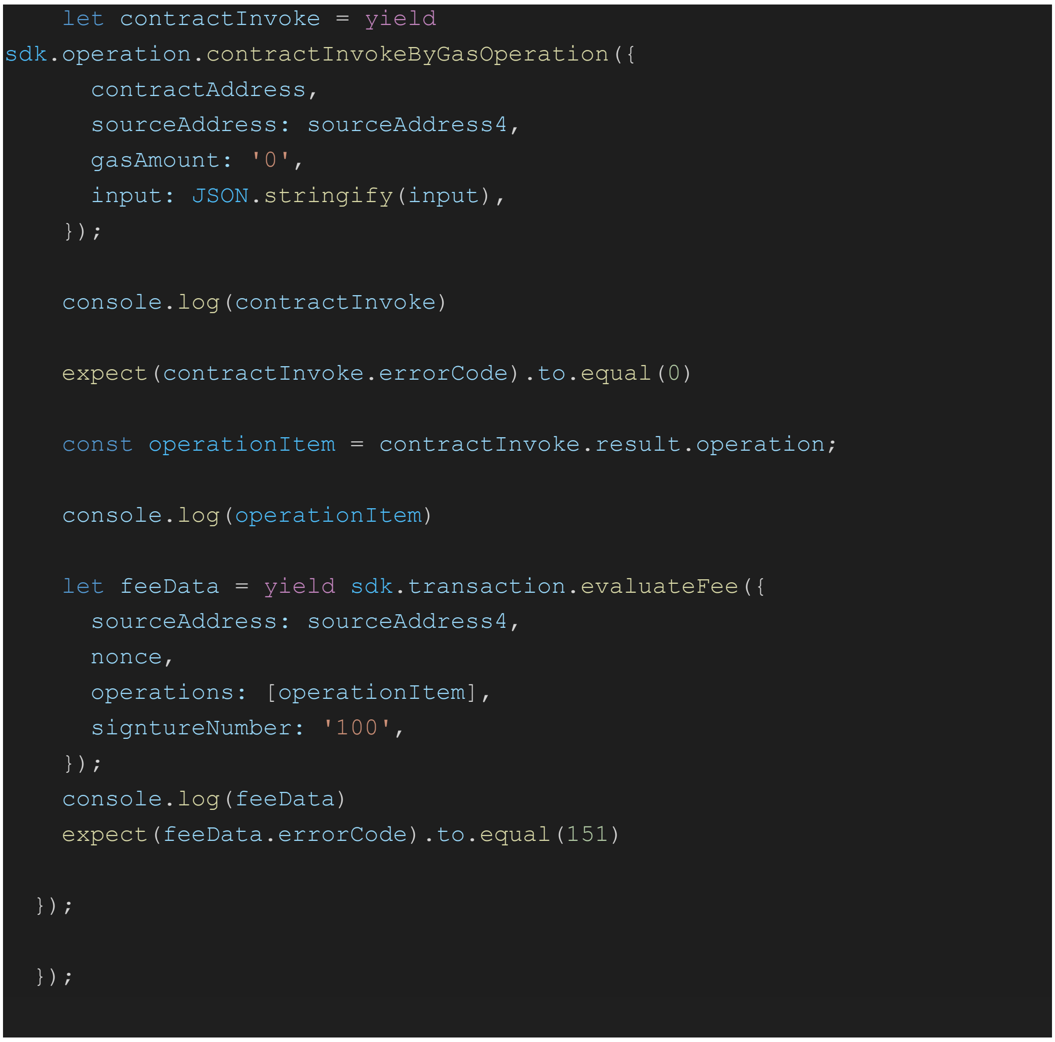
  it('validator apply amount != 0 check', function* () {

    const nonceResult = yield sdk.account.getNonce(sourceAddress4);

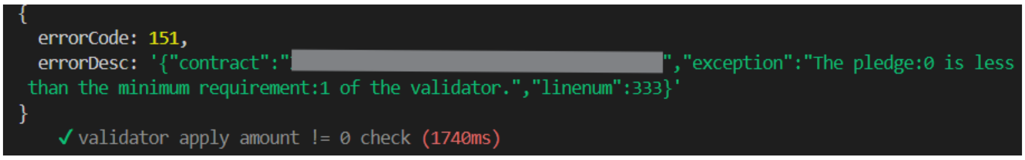
    expect(nonceResult.errorCode).to.equal(0)

    let nonce = nonceResult.result.nonce;
    nonce = new BigNumber(nonce).plus(1).toString(10);

    /*
     Specify the input parameters for invoking contract
     */
    let input = {
      "method": "apply",
      "params": {
        "role": "validator",
        "ratio": 0,
        "node": "sourceAddress4"
      }
    }
</pre></td></tr>
</table>



```
    let contractInvoke = yield
sdk.operation.contractInvokeByGasOperation({
      contractAddress,
      sourceAddress: sourceAddress4,
      gasAmount: '0',
      input: JSON.stringify(input),
    });

    console.log(contractInvoke)

    expect(contractInvoke.errorCode).to.equal(0)

    const operationItem = contractInvoke.result.operation;

    console.log(operationItem)

    let feeData = yield sdk.transaction.evaluateFee({
      sourceAddress: sourceAddress4,
      nonce,
      operations: [operationItem],
      signtureNumber: '100',
    });
    console.log(feeData)
    expect(feeData.errorCode).to.equal(151)

  });

  });
```

Expected Behavior	The gas amount being equal to 0 to apply for a validator role raises an error. 

```
{
  errorCode: 151,
  errorDesc: '{"contract":"[redacted]","exception":"The pledge:0 is less
 than the minimum requirement:1 of the validator.","linenum":333}'
}
    ✓ validator apply amount != 0 check (1740ms)
```

4.17 Validator apply check

Result	**PASS**
Function	Apply and Approve
Objective	Checks the process of applying for a validator role. It invokes the "apply" function with the specified parameters and verifies the result.

Test Case

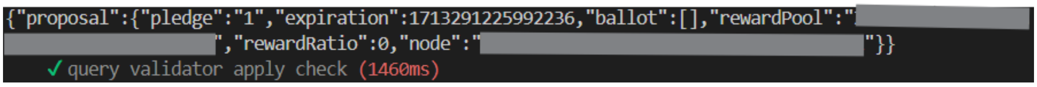
```
// committee - sourceAddress
describe('Apply and Approve test 9', async function () {
  this.timeout(30000);

  it('validator apply check', function* () {

    const nonceResult = yield sdk.account.getNonce(sourceAddress3);

    expect(nonceResult.errorCode).to.equal(0)

    let nonce = nonceResult.result.nonce;
    nonce = new BigNumber(nonce).plus(1).toString(10);

    /*
     Specify the input parameters for invoking contract
     */
    let input = {
      "method": "apply",
      "params": {
        "role": "validator",
        "ratio": 0,
        "node": "sourceAddress3"
      }
    }

    let contractInvoke = yield
sdk.operation.contractInvokeByGasOperation({
      contractAddress,
      sourceAddress: sourceAddress3,
      gasAmount: '1',
      input: JSON.stringify(input),
    });

    console.log(contractInvoke)

    expect(contractInvoke.errorCode).to.equal(0)

    const operationItem = contractInvoke.result.operation;

    console.log(operationItem)

    let feeData = yield sdk.transaction.evaluateFee({
      sourceAddress: sourceAddress3,
      nonce,
      operations: [operationItem],
      signtureNumber: '100',
    });
    console.log(feeData)
    expect(feeData.errorCode).to.equal(0)

    let feeLimit = feeData.result.feeLimit;
    let gasPrice = feeData.result.gasPrice;
```

	see code below

```
    console.log("gasPrice", gasPrice);
    console.log("feeLimit", feeLimit);

    const blobInfo = sdk.transaction.buildBlob({
      sourceAddress: sourceAddress3,
      gasPrice: gasPrice,
      feeLimit: feeLimit,
      nonce: nonce,
      operations: [operationItem],
    });

    console.log(blobInfo);
    expect(blobInfo.errorCode).to.equal(0)

    const signed = sdk.transaction.sign({
      privateKeys: [privateKey3],
      blob: blobInfo.result.transactionBlob
    })

    console.log(signed)
    expect(signed.errorCode).to.equal(0)

    let submitted = yield sdk.transaction.submit({
      signature: signed.result.signatures,
      blob: blobInfo.result.transactionBlob
    })

    console.log(submitted)
    expect(submitted.errorCode).to.equal(0)

    let info = null;
    for (let i = 0; i < 10; i++) {
      console.log("Getting the transaction history (attempt " + (i +
1).toString() + ")...")
      info = yield sdk.transaction.getInfo(submitted.result.hash)
      if (info.errorCode === 0) {
        break;
      }
      sleep(2000);
    }

    expect(info.errorCode).to.equal(0)

  });

  it('query validator apply check', async () => {

    let data = await sdk.contract.call({
      optType: 2,
      contractAddress: contractAddress,
      input: JSON.stringify({
        method: 'getProposal',
           params: {
             operate: 'apply',
             item: 'validator',
             address: 'sourceAddress3'
           }
      }),
    });
    console.log(data)
    console.log(data.result.query_rets[0].result.value);
    expect(data.errorCode).to.equal(0);

expect(data.result.query_rets[0].result.value).to.not.equal('{"proposal"
:false}');
  });


  });
```

Expected Behavior

As "proposal" is not false, the user has successfully applied to be a validator.



```
{"proposal":{"pledge":"1","expiration":1713291225992236,"ballot":[],"rewardPool":"[redacted]","rewardRatio":0,"node":"[redacted]"}}
    ✓ query validator apply check (1460ms)
```

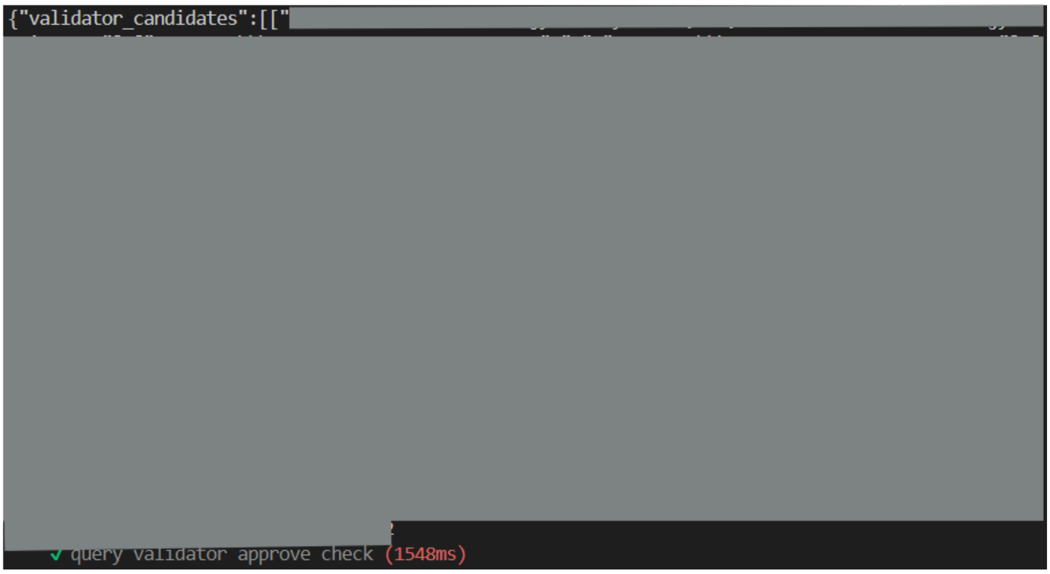
4.18 Validator approve check

Result	**PASS**
Function	Apply and Approve
Objective	Checks the process of approving a validator role proposal. It invokes the "approve" function with the specified parameters and verifies the result.

Test Case

```
// committee - sourceAddress; validator candidate - sourceAddress3
describe('Apply and Approve test 10', async function () {
  this.timeout(30000);

  it('validator approve check', function* () {

    const nonceResult = yield sdk.account.getNonce(sourceAddress);

    expect(nonceResult.errorCode).to.equal(0)

    let nonce = nonceResult.result.nonce;
    nonce = new BigNumber(nonce).plus(1).toString(10);

    /*
```

```
     Specify the input parameters for invoking contract
     */
    let input = {
      "method": "approve",
      "params": {
        "operate": "apply",
        "item": "validator",
        "address":  "sourceAddress3"
      }
    }

    let contractInvoke = yield
sdk.operation.contractInvokeByGasOperation({
      contractAddress,
      sourceAddress: sourceAddress,
      gasAmount: '0',
      input: JSON.stringify(input),
    });

    console.log(contractInvoke)

    expect(contractInvoke.errorCode).to.equal(0)

    const operationItem = contractInvoke.result.operation;

    console.log(operationItem)

    let feeData = yield sdk.transaction.evaluateFee({
      sourceAddress: sourceAddress,
      nonce,
      operations: [operationItem],
      signtureNumber: '100',
    });
    console.log(feeData)
    expect(feeData.errorCode).to.equal(0)

    let feeLimit = feeData.result.feeLimit;
    let gasPrice = feeData.result.gasPrice;

    console.log("gasPrice", gasPrice);
    console.log("feeLimit", feeLimit);

    const blobInfo = sdk.transaction.buildBlob({
      sourceAddress: sourceAddress,
      gasPrice: gasPrice,
      feeLimit: feeLimit,
      nonce: nonce,
      operations: [operationItem],
    });

    console.log(blobInfo);
    expect(blobInfo.errorCode).to.equal(0)
    const signed = sdk.transaction.sign({
      privateKeys: [privateKey],
      blob: blobInfo.result.transactionBlob
    })

    console.log(signed)
    expect(signed.errorCode).to.equal(0)

    let submitted = yield sdk.transaction.submit({
      signature: signed.result.signatures,
      blob: blobInfo.result.transactionBlob
    })

    console.log(submitted)
    expect(submitted.errorCode).to.equal(0)

    let info = null;
    for (let i = 0; i < 10; i++) {
      console.log("Getting the transaction history (attempt " + (i +
1).toString() + ")...")
      info = yield sdk.transaction.getInfo(submitted.result.hash)
      if (info.errorCode === 0) {
        break;
      }
      sleep(2000);
    }

    expect(info.errorCode).to.equal(0)

  });

  it('query validator approve check', async () => {

    let data = await sdk.contract.call({
      optType: 2,
      contractAddress: contractAddress,
      input: JSON.stringify({
        method: 'getValidatorCandidates',
      }),
    });
    console.log(data)
    console.log(data.result.query_rets[0].result.value)

console.log(JSON.parse(data.result.query_rets[0].result.value).validator
_candidates[0][0]);
    expect(data.errorCode).to.equal(0);

expect(JSON.parse(data.result.query_rets[0].result.value).validator_cand
idates[0][0]).to.equal('sourceAddress3')
  });

  });
```

Expected Behavior	The getValidatorCandidates query returns the address of the newly approved validator candidate member along with the existing members.  `{"validator_candidates":[["` [redacted] `✓ query validator approve check (1548ms)`

4.19 Freeze call by non-member of validator_freeze_account[]

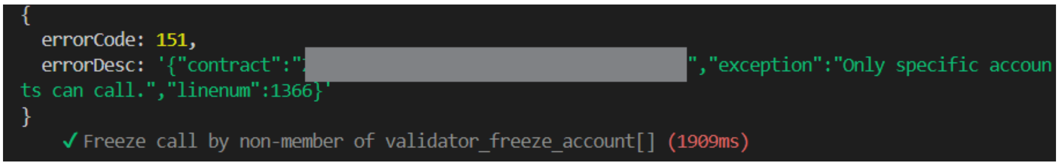
Result	**PASS**
Function	Freeze
Objective	Verifies that any member other that members of the “validator_freeze_account[]” can’t call the “freeze” function.

Test Case	`// validator_freeze_account - sourceAddress; Freeze - sourceAddress3`
	`describe('Freeze test 1', async function () {`
	`  this.timeout(30000);`
	
	`  it('Freeze call by non-member of validator_freeze_account[]', function* () {`
	
	`    const nonceResult = yield sdk.account.getNonce(sourceAddress2);`
	
	`    expect(nonceResult.errorCode).to.equal(0)`
	
	`    let nonce = nonceResult.result.nonce;`
	`    nonce = new BigNumber(nonce).plus(1).toString(10);`
	
	`    /*`
	`     Specify the input parameters for invoking contract`

```
   */
  let input = {
   "method": "setFreeze",
   "params": {
    "freeze": false,
    "validators": [
     "sourceAddress3"
    ]
   }
  }

  let contractInvoke = yield
sdk.operation.contractInvokeByGasOperation({
   contractAddress,
   sourceAddress: sourceAddress2,
   gasAmount: '0',
   input: JSON.stringify(input),
  });

  console.log(contractInvoke)

  expect(contractInvoke.errorCode).to.equal(0)

  const operationItem = contractInvoke.result.operation;

  console.log(operationItem)

  let feeData = yield sdk.transaction.evaluateFee({
   sourceAddress: sourceAddress2,
   nonce,
   operations: [operationItem],
   signtureNumber: '100',
  });
  console.log(feeData)
  expect(feeData.errorCode).to.equal(151)


 });

 });
```

Expected Behavior

A non-member of validator_freeze_account[] calling the "setFreeze" function raises an error.



```
{
  errorCode: 151,
  errorDesc: '{"contract":"[redacted]","exception":"Only specific accoun
ts can call.","linenum":1366}'
}
    ✓ Freeze call by non-member of validator_freeze_account[] (1909ms)
```

4.20 Freeze true check

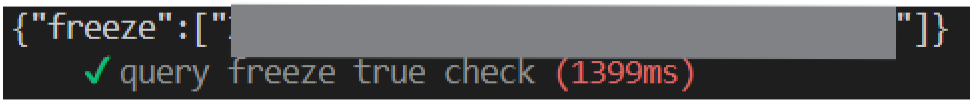
Result	**PASS**
Function	Freeze
Objective	Checks the process of freezing an account. It invokes the "freeze" function with the specified parameters and verifies the result.
Test Case	

```
// validator_freeze_account - sourceAddress; Freeze - sourceAddress3
describe('Freeze test 2', async function () {
  this.timeout(30000);

  it('Freeze true check', function* () {

    const nonceResult = yield sdk.account.getNonce(sourceAddress);

    expect(nonceResult.errorCode).to.equal(0)

    let nonce = nonceResult.result.nonce;
    nonce = new BigNumber(nonce).plus(1).toString(10);

    /*
     Specify the input parameters for invoking contract
     */
    let input = {
      "method": "setFreeze",
      "params": {
        "freeze": true,
        "validators": [
          "sourceAddress3"
        ]
      }
    }

    let contractInvoke = yield
sdk.operation.contractInvokeByGasOperation({
      contractAddress,
      sourceAddress: sourceAddress,
      gasAmount: '0',
      input: JSON.stringify(input),
    });

    console.log(contractInvoke)

    expect(contractInvoke.errorCode).to.equal(0)
    const operationItem = contractInvoke.result.operation;

    console.log(operationItem)

    let feeData = yield sdk.transaction.evaluateFee({
      sourceAddress: sourceAddress,
      nonce,
      operations: [operationItem],
      signtureNumber: '100',
    });
    console.log(feeData)
    expect(feeData.errorCode).to.equal(0)

    let feeLimit = feeData.result.feeLimit;
    let gasPrice = feeData.result.gasPrice;

    console.log("gasPrice", gasPrice);
    console.log("feeLimit", feeLimit);

    const blobInfo = sdk.transaction.buildBlob({
      sourceAddress: sourceAddress,
      gasPrice: gasPrice,
      feeLimit: feeLimit,
      nonce: nonce,
      operations: [operationItem],
    });

    console.log(blobInfo);
    expect(blobInfo.errorCode).to.equal(0)

    const signed = sdk.transaction.sign({
      privateKeys: [privateKey],
      blob: blobInfo.result.transactionBlob
    })

    console.log(signed)
    expect(signed.errorCode).to.equal(0)

    let submitted = yield sdk.transaction.submit({
      signature: signed.result.signatures,
      blob: blobInfo.result.transactionBlob
    })

    console.log(submitted)
    expect(submitted.errorCode).to.equal(0)

    let info = null;
    for (let i = 0; i < 10; i++) {
      console.log("Getting the transaction history (attempt " + (i +
1).toString() + ")...")
      info = yield sdk.transaction.getInfo(submitted.result.hash)
      if (info.errorCode === 0) {
        break;
```

```
   }
   sleep(2000);
  }

  expect(info.errorCode).to.equal(0)

 });

 it('query freeze true check', async () => {

  let data = await sdk.contract.call({
   optType: 2,
   contractAddress: contractAddress,
   input: JSON.stringify({
    method: 'getFreeze',
   }),
  });
  console.log(data)
  console.log(data.result.query_rets[0].result.value);
  expect(data.errorCode).to.equal(0);

expect(data.result.query_rets[0].result.value).to.equal('{"freeze":["sou
rceAddress3"]}')
 });


 });
```

Expected Behavior

The Freezed account appears in the getFreeze query output.



```
{"freeze":["[redacted]"]}
    ✓ query freeze true check (1399ms)
```

4.21 Freeze false check

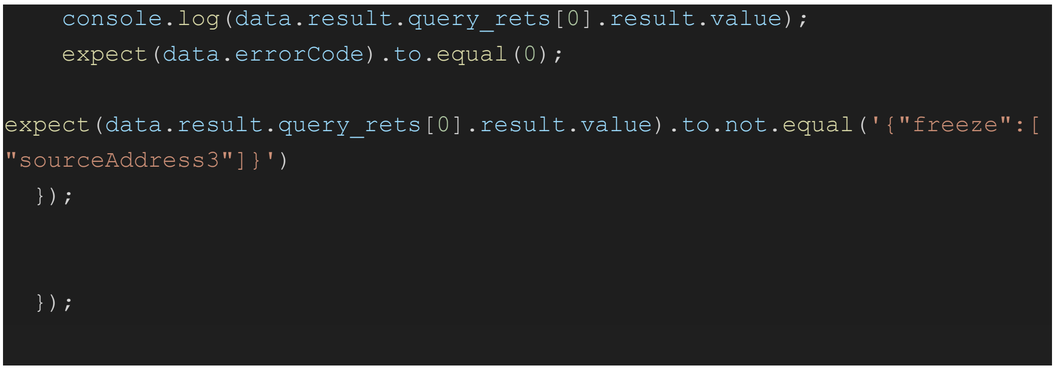
Result	**PASS**
Function	Freeze
Objective	Checks the process of un-freezing an account. It invokes the "freeze" function with the specified parameters and verifies the result.

Test Case

```
// validator_freeze_account - sourceAddress; Freeze - sourceAddress3
describe('Freeze test 3', async function () {
 this.timeout(30000);
```

```
it('Freeze false check', function* () {

  const nonceResult = yield sdk.account.getNonce(sourceAddress);

  expect(nonceResult.errorCode).to.equal(0)

  let nonce = nonceResult.result.nonce;
  nonce = new BigNumber(nonce).plus(1).toString(10);

  /*
   Specify the input parameters for invoking contract
   */
  let input = {
    "method": "setFreeze",
    "params": {
      "freeze": false,
      "validators": [
        "sourceAddress3"
      ]
    }
  }

  let contractInvoke = yield
sdk.operation.contractInvokeByGasOperation({
    contractAddress,
    sourceAddress: sourceAddress,
    gasAmount: '0',
    input: JSON.stringify(input),
  });

  console.log(contractInvoke)

  expect(contractInvoke.errorCode).to.equal(0)

  const operationItem = contractInvoke.result.operation;

  console.log(operationItem)

  let feeData = yield sdk.transaction.evaluateFee({
    sourceAddress: sourceAddress,
    nonce,
    operations: [operationItem],
    signtureNumber: '100',
  });
  console.log(feeData)
  expect(feeData.errorCode).to.equal(0)

  let feeLimit = feeData.result.feeLimit;
  let gasPrice = feeData.result.gasPrice;

  console.log("gasPrice", gasPrice);
  console.log("feeLimit", feeLimit);
```

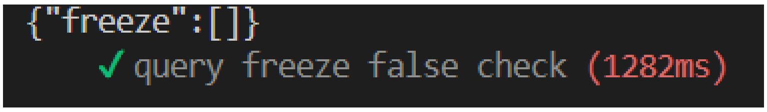
```
    const blobInfo = sdk.transaction.buildBlob({
      sourceAddress: sourceAddress,
      gasPrice: gasPrice,
      feeLimit: feeLimit,
      nonce: nonce,
      operations: [operationItem],
    });

    console.log(blobInfo);
    expect(blobInfo.errorCode).to.equal(0)

    const signed = sdk.transaction.sign({
      privateKeys: [privateKey],
      blob: blobInfo.result.transactionBlob
    })

    console.log(signed)
    expect(signed.errorCode).to.equal(0)

    let submitted = yield sdk.transaction.submit({
      signature: signed.result.signatures,
      blob: blobInfo.result.transactionBlob
    })

    console.log(submitted)
    expect(submitted.errorCode).to.equal(0)

    let info = null;
    for (let i = 0; i < 10; i++) {
      console.log("Getting the transaction history (attempt " + (i +
1).toString() + ")...")
      info = yield sdk.transaction.getInfo(submitted.result.hash)
      if (info.errorCode === 0) {
        break;
      }
      sleep(2000);
    }

    expect(info.errorCode).to.equal(0)

  });

  it('query freeze false check', async () => {

    let data = await sdk.contract.call({
      optType: 2,
      contractAddress: contractAddress,
      input: JSON.stringify({
        method: 'getFreeze',
      }),
    });
    console.log(data)
```



```
    console.log(data.result.query_rets[0].result.value);
    expect(data.errorCode).to.equal(0);

expect(data.result.query_rets[0].result.value).to.not.equal('{"freeze":[
"sourceAddress3"]}')
  });


  });
```

Expected Behavior

The un-Freezed account doesn't appear in the getFreeze query output.

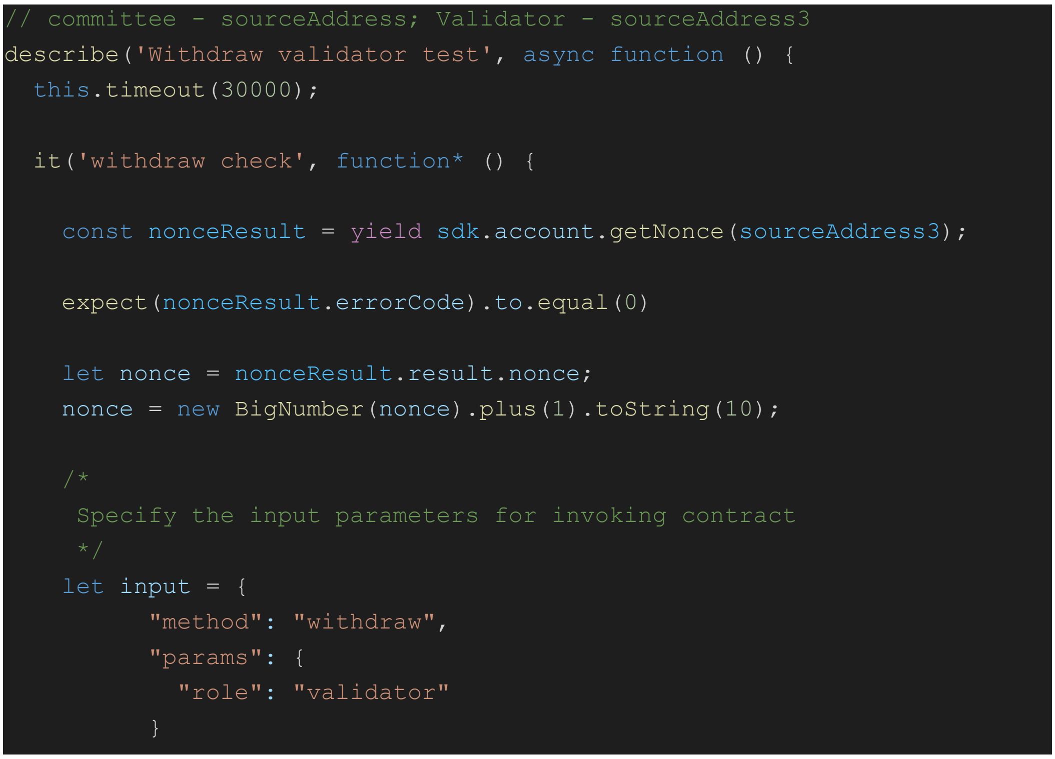


```
{"freeze":[]}
      ✓ query freeze false check (1282ms)
```

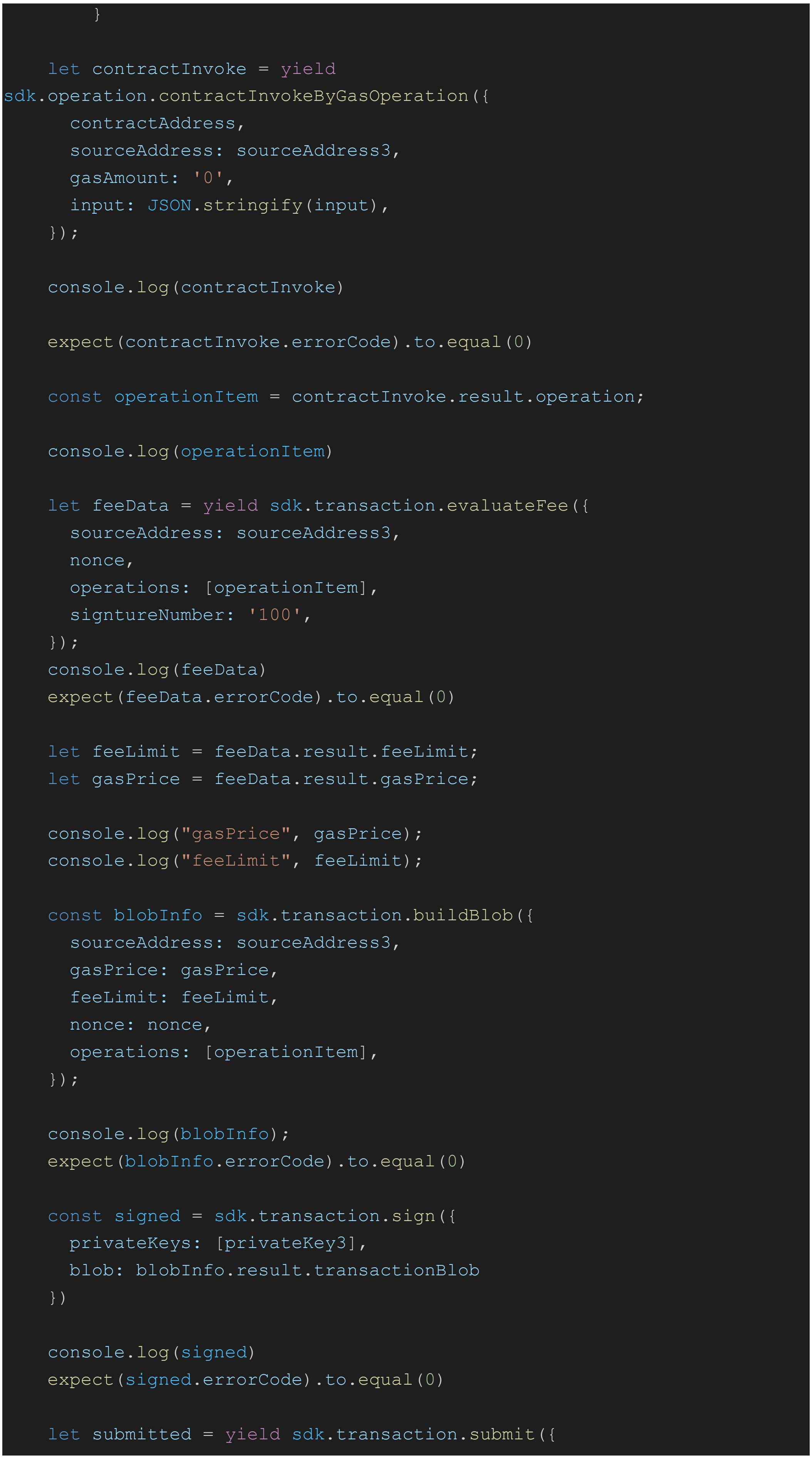
4.22 withdraw check

Result	**PASS**
Function	Withdraw
Objective	Checks the process of withdrawing a validator candidate from the smart contract. It invokes the "withdraw" function and verifies the result.

Test Case

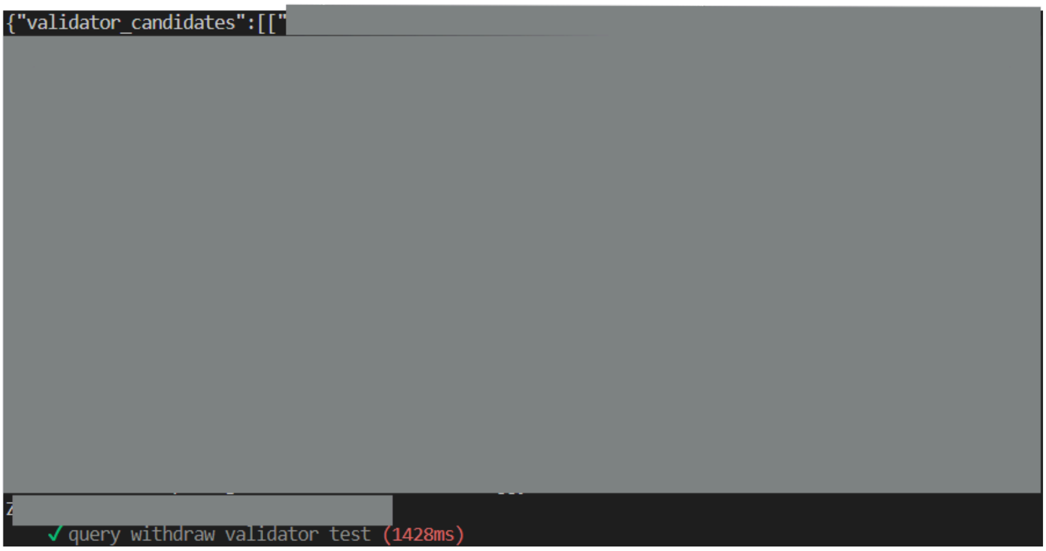


```
// committee - sourceAddress; Validator - sourceAddress3
describe('Withdraw validator test', async function () {
  this.timeout(30000);

  it('withdraw check', function* () {

    const nonceResult = yield sdk.account.getNonce(sourceAddress3);

    expect(nonceResult.errorCode).to.equal(0)

    let nonce = nonceResult.result.nonce;
    nonce = new BigNumber(nonce).plus(1).toString(10);

    /*
     Specify the input parameters for invoking contract
     */
    let input = {
          "method": "withdraw",
          "params": {
            "role": "validator"
          }
```



```
        }

    let contractInvoke = yield
sdk.operation.contractInvokeByGasOperation({
      contractAddress,
      sourceAddress: sourceAddress3,
      gasAmount: '0',
      input: JSON.stringify(input),
    });

    console.log(contractInvoke)

    expect(contractInvoke.errorCode).to.equal(0)

    const operationItem = contractInvoke.result.operation;

    console.log(operationItem)

    let feeData = yield sdk.transaction.evaluateFee({
      sourceAddress: sourceAddress3,
      nonce,
      operations: [operationItem],
      signtureNumber: '100',
    });
    console.log(feeData)
    expect(feeData.errorCode).to.equal(0)

    let feeLimit = feeData.result.feeLimit;
    let gasPrice = feeData.result.gasPrice;

    console.log("gasPrice", gasPrice);
    console.log("feeLimit", feeLimit);

    const blobInfo = sdk.transaction.buildBlob({
      sourceAddress: sourceAddress3,
      gasPrice: gasPrice,
      feeLimit: feeLimit,
      nonce: nonce,
      operations: [operationItem],
    });

    console.log(blobInfo);
    expect(blobInfo.errorCode).to.equal(0)

    const signed = sdk.transaction.sign({
      privateKeys: [privateKey3],
      blob: blobInfo.result.transactionBlob
    })

    console.log(signed)
    expect(signed.errorCode).to.equal(0)

    let submitted = yield sdk.transaction.submit({
      signature: signed.result.signatures,
      blob: blobInfo.result.transactionBlob
    })

    console.log(submitted)
    expect(submitted.errorCode).to.equal(0)

    let info = null;
    for (let i = 0; i < 10; i++) {
      console.log("Getting the transaction history (attempt " + (i +
1).toString() + ")...")
      info = yield sdk.transaction.getInfo(submitted.result.hash)
      if (info.errorCode === 0) {
        break;
      }
      sleep(2000);
    }

    expect(info.errorCode).to.equal(0)

  });

  it('query withdraw validator test', async () => {

    let data = await sdk.contract.call({
      optType: 2,
      contractAddress: contractAddress,
      input: JSON.stringify({
        method: 'getValidatorCandidates',
      }),
    });
    console.log(data)
    console.log(data.result.query_rets[0].result.value)

console.log(JSON.parse(data.result.query_rets[0].result.value).validator
_candidates[0][0]);
    expect(data.errorCode).to.equal(0);

expect(JSON.parse(data.result.query_rets[0].result.value).validator_cand
idates[0][0]).to.not.equal('sourceAddress3')
  });

  });
```

Expected Behavior	A validator withdraws and is no longer visible in the getValidatorCandidates query.

<table>
<tr><td></td><td><pre>{"validator_candidates":[["[redacted]
[redacted]
    ✓ query withdraw validator test (1428ms)</pre></td></tr>
</table>

4.23 fallback check

<table>
<tr><td><b>Result</b></td><td><b>PASS</b></td></tr>
<tr><td><b>Function</b></td><td>Fallback</td></tr>
<tr><td><b>Objective</b></td><td>Checks the functioning of the fallback function.</td></tr>
<tr><td><b>Test Case</b></td><td><pre>describe('fallback test', async function () {
  this.timeout(30000);

  it('fallback check', function* () {

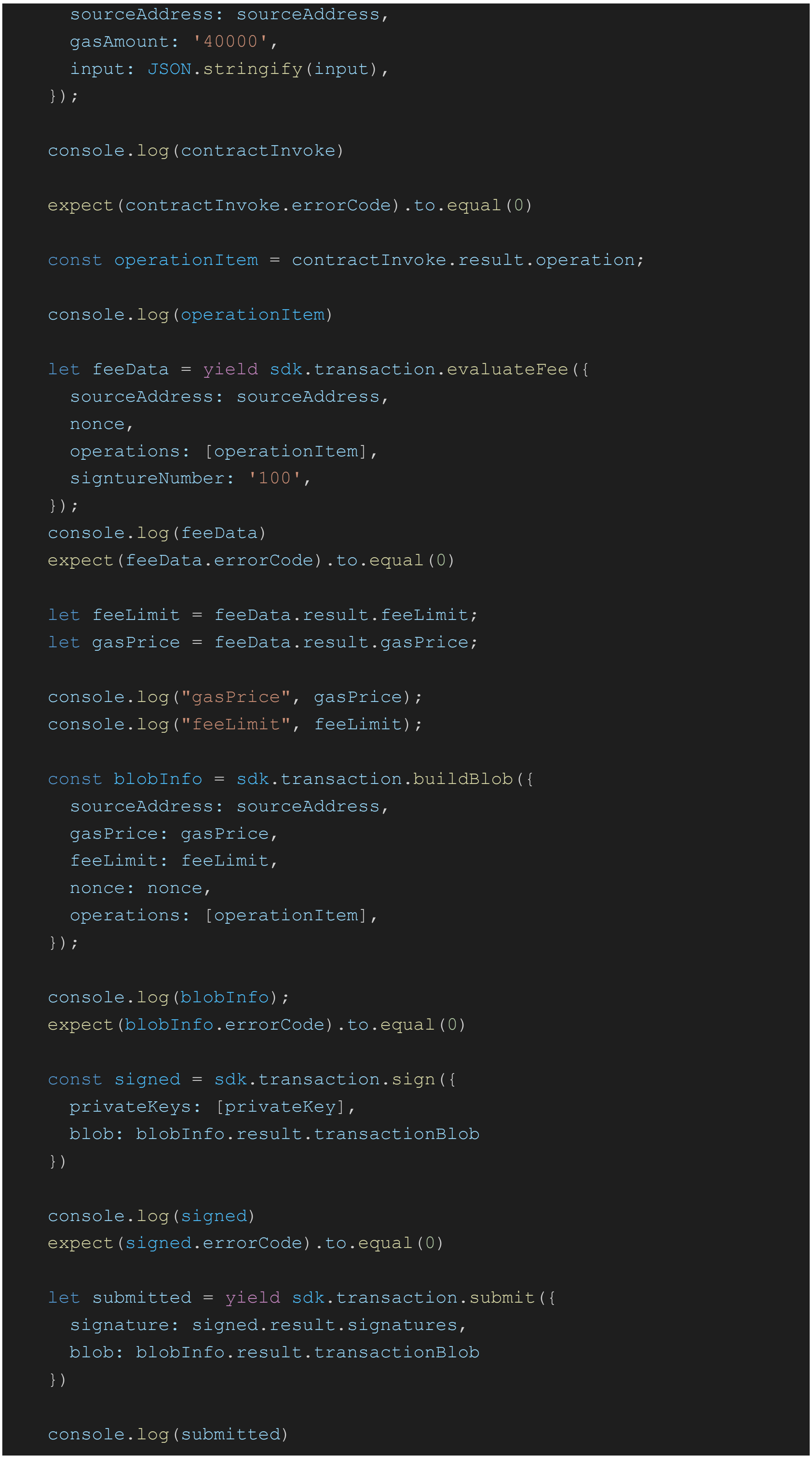
    const nonceResult = yield sdk.account.getNonce(sourceAddress);

    expect(nonceResult.errorCode).to.equal(0)

    let nonce = nonceResult.result.nonce;
    nonce = new BigNumber(nonce).plus(1).toString(10);

    /*
     Specify the input parameters for invoking contract
     */
    let input = {
          "method": "fallback"
        }

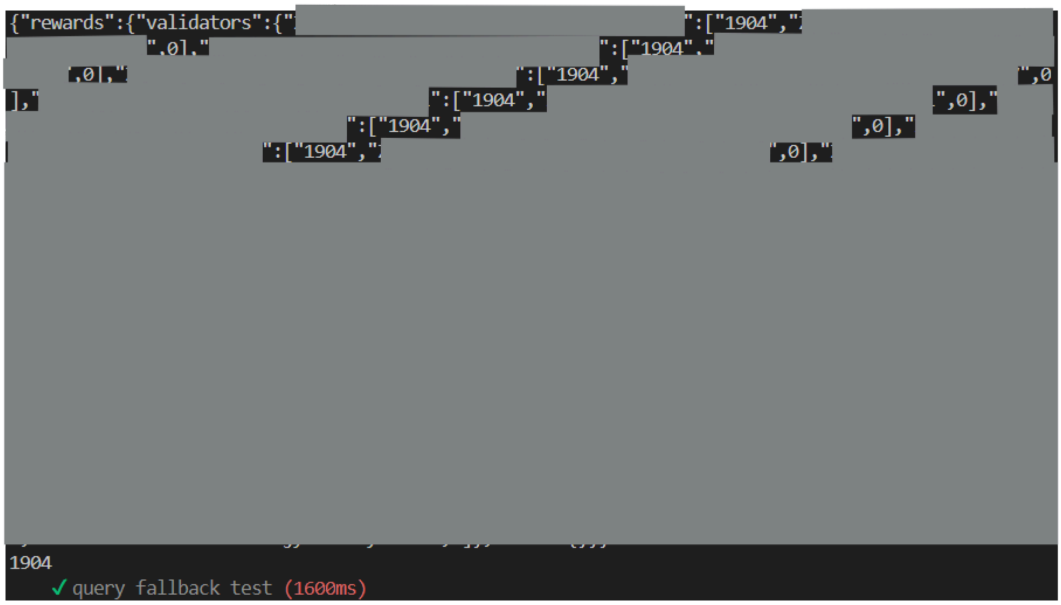
    let contractInvoke = yield
sdk.operation.contractInvokeByGasOperation({
      contractAddress,</pre></td></tr>
</table>



```
  sourceAddress: sourceAddress,
  gasAmount: '40000',
  input: JSON.stringify(input),
});

console.log(contractInvoke)

expect(contractInvoke.errorCode).to.equal(0)

const operationItem = contractInvoke.result.operation;

console.log(operationItem)

let feeData = yield sdk.transaction.evaluateFee({
  sourceAddress: sourceAddress,
  nonce,
  operations: [operationItem],
  signtureNumber: '100',
});
console.log(feeData)
expect(feeData.errorCode).to.equal(0)

let feeLimit = feeData.result.feeLimit;
let gasPrice = feeData.result.gasPrice;

console.log("gasPrice", gasPrice);
console.log("feeLimit", feeLimit);

const blobInfo = sdk.transaction.buildBlob({
  sourceAddress: sourceAddress,
  gasPrice: gasPrice,
  feeLimit: feeLimit,
  nonce: nonce,
  operations: [operationItem],
});

console.log(blobInfo);
expect(blobInfo.errorCode).to.equal(0)

const signed = sdk.transaction.sign({
  privateKeys: [privateKey],
  blob: blobInfo.result.transactionBlob
})

console.log(signed)
expect(signed.errorCode).to.equal(0)

let submitted = yield sdk.transaction.submit({
  signature: signed.result.signatures,
  blob: blobInfo.result.transactionBlob
})

console.log(submitted)
```

```
    expect(submitted.errorCode).to.equal(0)

    let info = null;
    for (let i = 0; i < 10; i++) {
      console.log("Getting the transaction history (attempt " + (i +
1).toString() + ")...")
      info = yield sdk.transaction.getInfo(submitted.result.hash)
      if (info.errorCode === 0) {
        break;
      }
      sleep(2000);
    }

    expect(info.errorCode).to.equal(0)

  });

  it('query fallback test', async () => {

    let data = await sdk.contract.call({
      optType: 2,
      contractAddress: contractAddress,
      input: JSON.stringify({
        method: 'getRewardDistribute',
      }),
    });
    console.log(data)
    console.log(data.result.query_rets[0].result.value)

console.log(Object.values(JSON.parse(data.result.query_rets[0].result.va
lue).rewards.validators)[0][0]);
    expect(data.errorCode).to.equal(0);

expect(Object.values(JSON.parse(data.result.query_rets[0].result.value).
rewards.validators)[0][0]).to.not.equal(0)
  });

  });
```

Expected Behavior	The balance in the validators accounts is not 0.

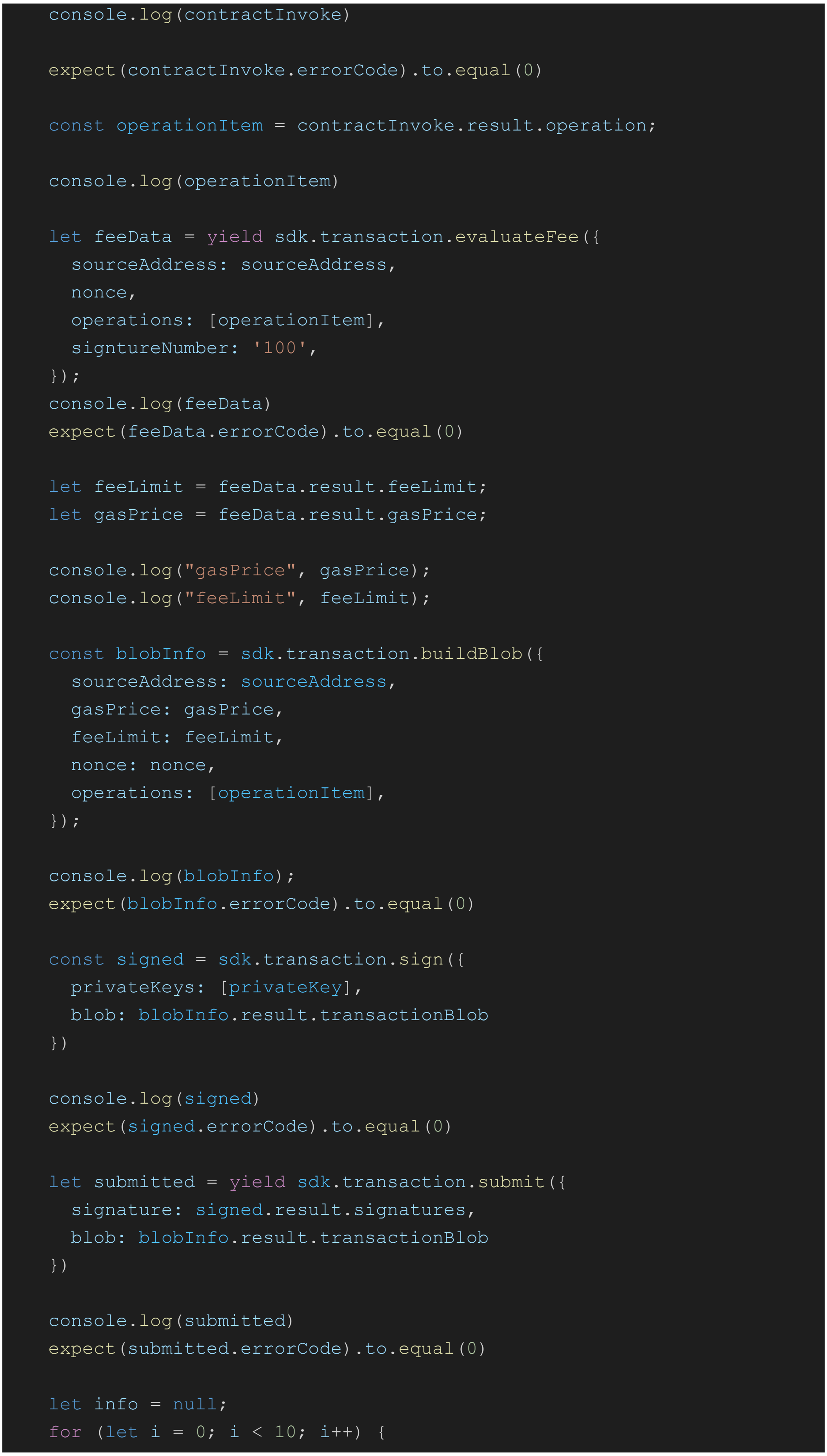


```
{"rewards":{"validators":{"[redacted]":["1904","[redacted]
[redacted]",0],"[redacted]":["1904","[redacted]
[redacted],0],"[redacted]":["1904","[redacted]",0
],"[redacted]":["1904","[redacted]",0],"
[redacted]":["1904","[redacted]",0],"
[redacted]":["1904","[redacted],0],"
[redacted]
1904
    ✓ query fallback test (1600ms)
```

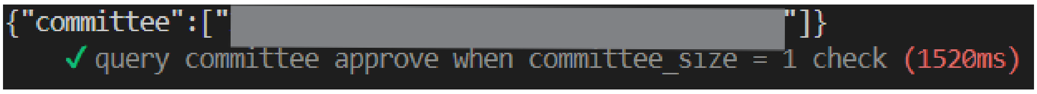
4.24 committee apply committee_size = 1

Result	**PASS**
Function	Apply and Approve
Objective	Verifies that the committee doesn’t accept new committee members when the “committee_size” is equal to the present size of the committee.
Test Case	

```
// committee - sourceAddress; committee_size == 1
describe('Apply and Approve test 11', async function () {
  this.timeout(30000);

  it('committee apply committee_size = 1', function* () {

    const nonceResult = yield sdk.account.getNonce(sourceAddress3);

    expect(nonceResult.errorCode).to.equal(0)

    let nonce = nonceResult.result.nonce;
    nonce = new BigNumber(nonce).plus(1).toString(10);

    /*
     Specify the input parameters for invoking contract
     */
    let input = {
      "method": "apply",
      "params": {
        "role": "committee",
        "ratio": 0,
        "node": "sourceAddress3"
      }
    }

    let contractInvoke = yield
sdk.operation.contractInvokeByGasOperation({
      contractAddress,
      sourceAddress: sourceAddress3,
      gasAmount: '0',
      input: JSON.stringify(input),
    });

    console.log(contractInvoke)

    expect(contractInvoke.errorCode).to.equal(0)

    const operationItem = contractInvoke.result.operation;

    console.log(operationItem)

    let feeData = yield sdk.transaction.evaluateFee({
      sourceAddress: sourceAddress3,
      nonce,
      operations: [operationItem],
      signtureNumber: '100',
    });
    console.log(feeData)
    expect(feeData.errorCode).to.equal(0)

    let feeLimit = feeData.result.feeLimit;
    let gasPrice = feeData.result.gasPrice;

    console.log("gasPrice", gasPrice);
    console.log("feeLimit", feeLimit);

    const blobInfo = sdk.transaction.buildBlob({
      sourceAddress: sourceAddress3,
      gasPrice: gasPrice,
      feeLimit: feeLimit,
      nonce: nonce,
      operations: [operationItem],
    });

    console.log(blobInfo);
    expect(blobInfo.errorCode).to.equal(0)

    const signed = sdk.transaction.sign({
      privateKeys: [privateKey3],
      blob: blobInfo.result.transactionBlob
    })

    console.log(signed)
    expect(signed.errorCode).to.equal(0)
    let submitted = yield sdk.transaction.submit({
      signature: signed.result.signatures,
      blob: blobInfo.result.transactionBlob
    })

    console.log(submitted)
    expect(submitted.errorCode).to.equal(0)

    let info = null;
    for (let i = 0; i < 10; i++) {
      console.log("Getting the transaction history (attempt " + (i +
1).toString() + ")...")
      info = yield sdk.transaction.getInfo(submitted.result.hash)
      if (info.errorCode === 0) {
        break;
      }
      sleep(2000);
    }

    expect(info.errorCode).to.equal(0)

  });

  it('committee approve when committee_size = 1 check', function* () {

    const nonceResult = yield sdk.account.getNonce(sourceAddress);

    expect(nonceResult.errorCode).to.equal(0)

    let nonce = nonceResult.result.nonce;
    nonce = new BigNumber(nonce).plus(1).toString(10);

    /*
     Specify the input parameters for invoking contract
     */
    let input = {
      "method": "approve",
      "params": {
        "operate": "apply",
        "item": "committee",
        "address":  "sourceAddress3"
      }
    }

    let contractInvoke = yield
sdk.operation.contractInvokeByGasOperation({
      contractAddress,
      sourceAddress: sourceAddress,
      gasAmount: '0',
      input: JSON.stringify(input),
    });
```



```
console.log(contractInvoke)

expect(contractInvoke.errorCode).to.equal(0)

const operationItem = contractInvoke.result.operation;

console.log(operationItem)

let feeData = yield sdk.transaction.evaluateFee({
  sourceAddress: sourceAddress,
  nonce,
  operations: [operationItem],
  signtureNumber: '100',
});
console.log(feeData)
expect(feeData.errorCode).to.equal(0)

let feeLimit = feeData.result.feeLimit;
let gasPrice = feeData.result.gasPrice;

console.log("gasPrice", gasPrice);
console.log("feeLimit", feeLimit);

const blobInfo = sdk.transaction.buildBlob({
  sourceAddress: sourceAddress,
  gasPrice: gasPrice,
  feeLimit: feeLimit,
  nonce: nonce,
  operations: [operationItem],
});

console.log(blobInfo);
expect(blobInfo.errorCode).to.equal(0)

const signed = sdk.transaction.sign({
  privateKeys: [privateKey],
  blob: blobInfo.result.transactionBlob
})

console.log(signed)
expect(signed.errorCode).to.equal(0)

let submitted = yield sdk.transaction.submit({
  signature: signed.result.signatures,
  blob: blobInfo.result.transactionBlob
})

console.log(submitted)
expect(submitted.errorCode).to.equal(0)

let info = null;
for (let i = 0; i < 10; i++) {
      console.log("Getting the transaction history (attempt " + (i +
1).toString() + ")...")
      info = yield sdk.transaction.getInfo(submitted.result.hash)
      if (info.errorCode === 0) {
        break;
      }
      sleep(2000);
    }

    expect(info.errorCode).to.equal(0)

  });

  it('query committee approve when committee_size = 1 check', async ()
=> {

    let data = await sdk.contract.call({
      optType: 2,
      contractAddress: contractAddress,
      input: JSON.stringify({
        method: 'getCommittee',
      }),
    });
    console.log(data)
    console.log(data.result.query_rets[0].result.value);
    expect(data.errorCode).to.equal(0);

expect(data.result.query_rets[0].result.value).to.equal('{"committee":["
sourceAddress"]}')
  });

  });
```

Expected Behavior	Any new committee member is not accepted into the contract as the committee_size has exceeded and thus only the existing committee members' names appear in the getCommittee function.  {"committee":["[redacted]"]} ✓ query committee approve when committee_size = 1 check (1520ms)

4.25 Inconsistent Proposal Retrieval Functionality

Result	**Informational**
Objective	The proposal retrieval functionality within the smart contract exhibits inconsistent behavior across different input scenarios when attempting to retrieve proposals. Despite various input conditions - including unsubmitted addresses, invalid address formats, invalid item parameters, and incorrect

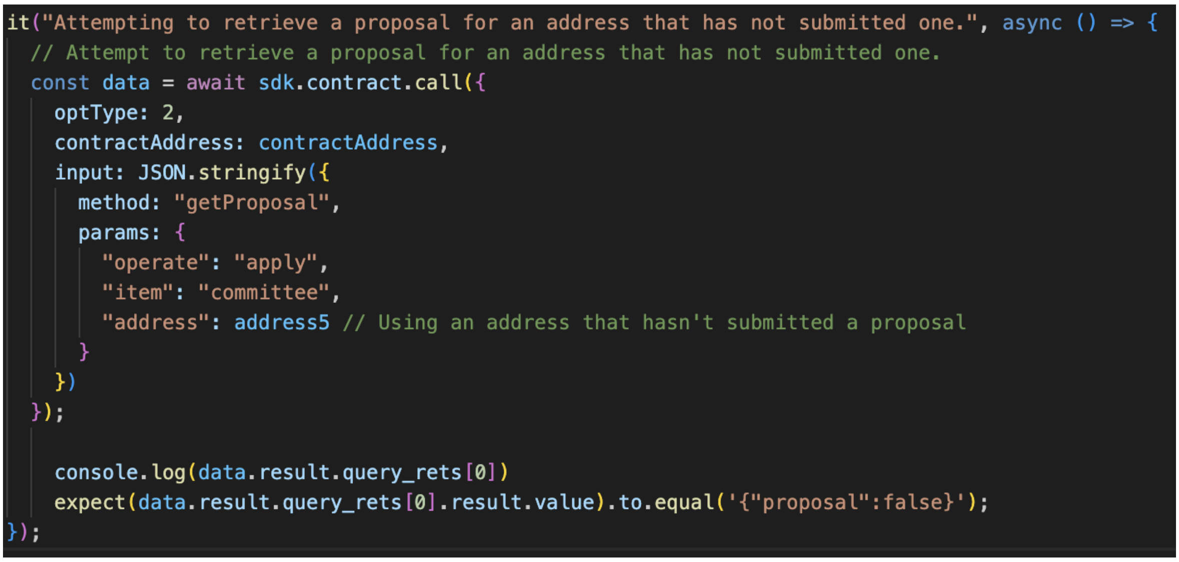
<table>
<tr>
<td></td>
<td>operation types - the contract consistently raises a response indicating no proposal found ({"proposal":false}). This uniform response suggests a lack of proper input validation and conditional handling within the getProposal method, potentially masking different underlying issues with a generic output.</td>
</tr>
<tr>
<td><b>Observation</b></td>
<td>

The provided test cases illustrate attempts to retrieve proposals under multiple conditions:

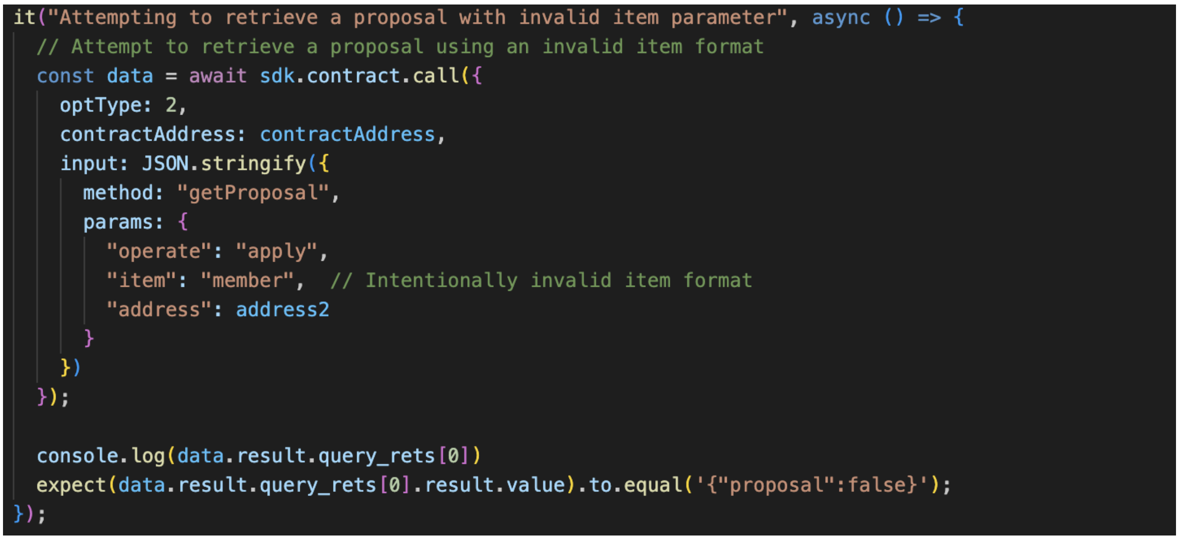
- Utilization using an address that has not submitted a proposal.
- Use of an invalid address format.
- Specification of an invalid item parameter.
- Requesting with a valid item format, but for an address that did not submit any proposal.
- Utilization of an invalid operation functionality.

In all instances, the contract's response does not differentiate between the nature of the input error or the absence of a proposal, suggesting that the contract lacks detailed input validation or specific error handling logic. This behavior can obfuscate the actual issue at hand, making it difficult for users to understand the failure reason and for developers to diagnose and rectify potential bugs.

</td>
</tr>
<tr>
<td><b>Test Case</b></td>
<td>

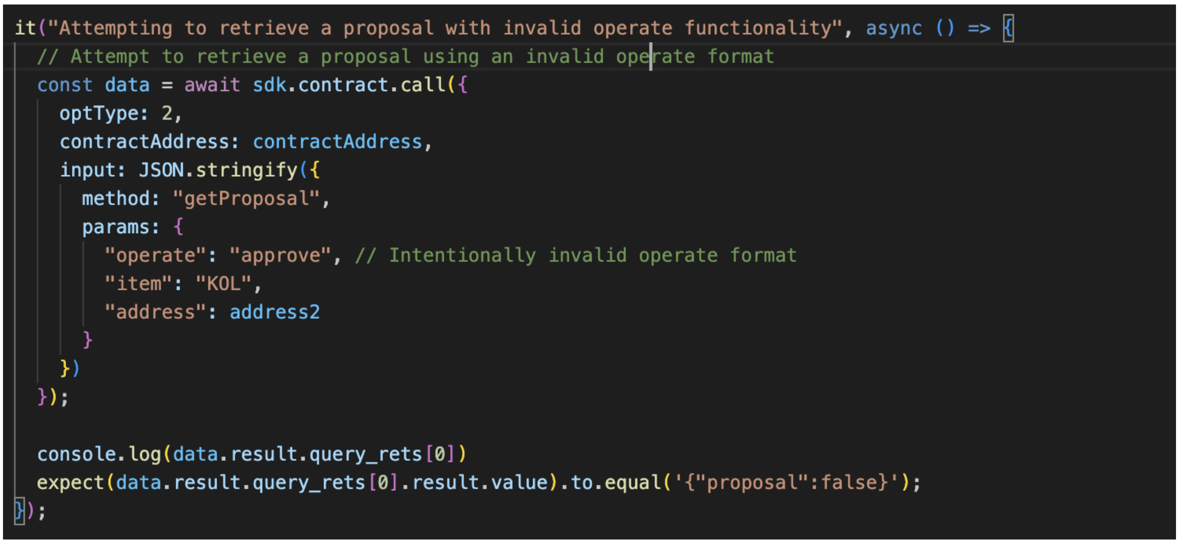


```
it("Attempting to retrieve a proposal for an address that has not submitted one.", async () => {
  // Attempt to retrieve a proposal for an address that has not submitted one.
  const data = await sdk.contract.call({
    optType: 2,
    contractAddress: contractAddress,
    input: JSON.stringify({
      method: "getProposal",
      params: {
        "operate": "apply",
        "item": "committee",
        "address": address5 // Using an address that hasn't submitted a proposal
      }
    })
  });

    console.log(data.result.query_rets[0])
    expect(data.result.query_rets[0].result.value).to.equal('{"proposal":false}');
});
```

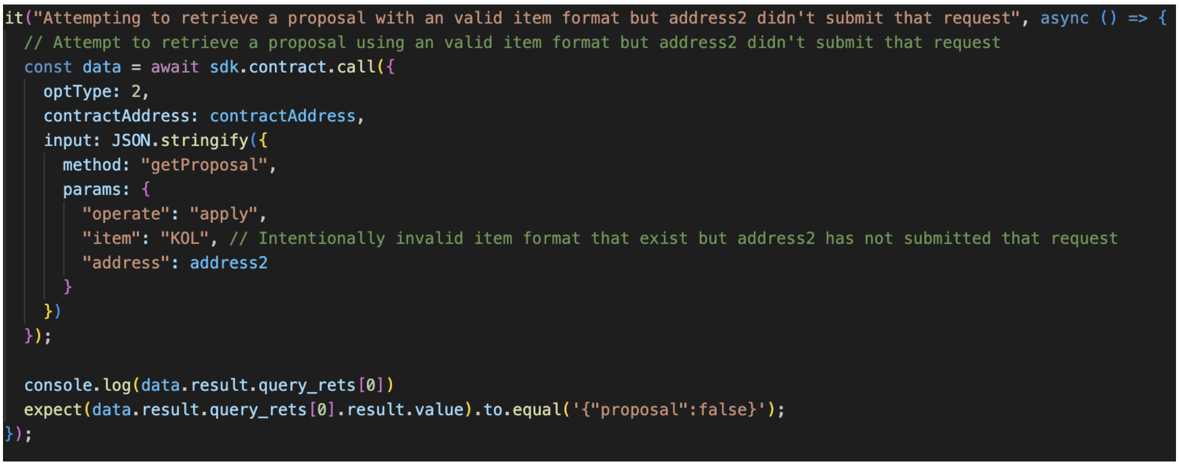


```
it("Attempting to retrieve a proposal with invalid item parameter", async () => {
  // Attempt to retrieve a proposal using an invalid item format
  const data = await sdk.contract.call({
    optType: 2,
    contractAddress: contractAddress,
    input: JSON.stringify({
      method: "getProposal",
      params: {
        "operate": "apply",
        "item": "member",  // Intentionally invalid item format
        "address": address2
      }
    })
  });

  console.log(data.result.query_rets[0])
  expect(data.result.query_rets[0].result.value).to.equal('{"proposal":false}');
});
```

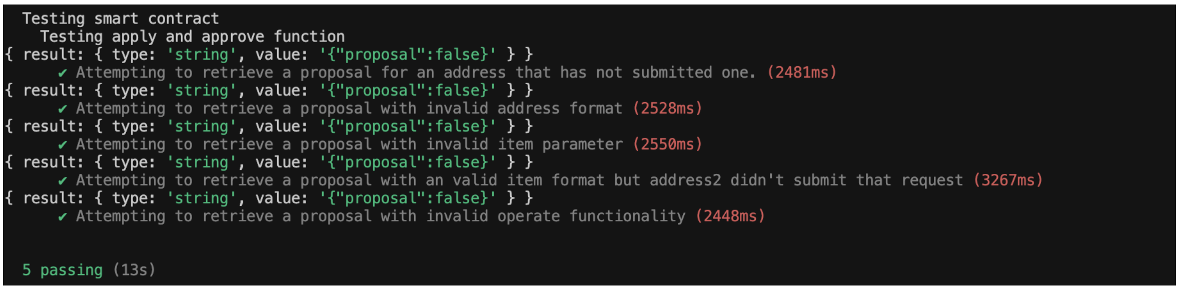
</td>
</tr>
</table>



```
it("Attempting to retrieve a proposal with invalid operate functionality", async () => {
  // Attempt to retrieve a proposal using an invalid operate format
  const data = await sdk.contract.call({
    optType: 2,
    contractAddress: contractAddress,
    input: JSON.stringify({
      method: "getProposal",
      params: {
        "operate": "approve", // Intentionally invalid operate format
        "item": "KOL",
        "address": address2
      }
    })
  });

  console.log(data.result.query_rets[0])
  expect(data.result.query_rets[0].result.value).to.equal('{"proposal":false}');
});
```



```
it("Attempting to retrieve a proposal with an valid item format but address2 didn't submit that request", async () => {
  // Attempt to retrieve a proposal using an valid item format but address2 didn't submit that request
  const data = await sdk.contract.call({
    optType: 2,
    contractAddress: contractAddress,
    input: JSON.stringify({
      method: "getProposal",
      params: {
        "operate": "apply",
        "item": "KOL", // Intentionally invalid item format that exist but address2 has not submitted that request
        "address": address2
      }
    })
  });

  console.log(data.result.query_rets[0])
  expect(data.result.query_rets[0].result.value).to.equal('{"proposal":false}');
});
```



```
  Testing smart contract
    Testing apply and approve function
{ result: { type: 'string', value: '{"proposal":false}' } }
      ✔ Attempting to retrieve a proposal for an address that has not submitted one. (2481ms)
{ result: { type: 'string', value: '{"proposal":false}' } }
      ✔ Attempting to retrieve a proposal with invalid address format (2528ms)
{ result: { type: 'string', value: '{"proposal":false}' } }
      ✔ Attempting to retrieve a proposal with invalid item parameter (2550ms)
{ result: { type: 'string', value: '{"proposal":false}' } }
      ✔ Attempting to retrieve a proposal with an valid item format but address2 didn't submit that request (3267ms)
{ result: { type: 'string', value: '{"proposal":false}' } }
      ✔ Attempting to retrieve a proposal with invalid operate functionality (2448ms)


  5 passing (13s)
```

Remediation	Enhance the getProposal method to include comprehensive input validation and conditional checks. Ensure that distinct error messages or codes are returned for different failure scenarios, invalid addresses, unsubmitted proposals, or incorrect parameters. Implementing detailed checks and responses will improve the contract's usability and debuggability, helping users understand the failure context and developers to identify issues more effectively.

5.0 Auditing Approach and Methodologies Applied

The Javascript smart contract was audited in a comprehensive approach to ensure the highest level of security and reliability. Careful attention was given to the following key areas to ensure the overall quality of code:

- **Code quality and structure**: We conducted a detailed review of the codebase to identify any potential issues related to code structure, readability, and maintainability.
- **Security vulnerabilities**: Our team used manual techniques to identify any potential security vulnerabilities that could be exploited by attackers. This involved a thorough analysis of the code to identify any potential weaknesses, such as buffer overflows, injection vulnerabilities, Signatures, and deprecated functions.

Our audit team followed OWASP and community security guidelines for this audit. As a result, we were able to identify potential issues and provide recommendations to improve the smart contract security and performance.

Throughout the audit of the smart contract, our team placed great emphasis on ensuring the overall quality of the code and the use of industry best practices. We meticulously reviewed the codebase to ensure that it was thoroughly documented and that all comments and logic aligned with the intended behavior. Our approach to the audit was comprehensive, methodical, and aimed at ensuring that the smart contract was secure, reliable, and optimized for performance.

5.1 Code Review / Manual Analysis

Our team conducted a manual analysis of the Javascript smart contracts to identify new vulnerabilities or to verify vulnerabilities found during manual analysis. We carefully analyzed every line of code and made sure that all instructions provided during the onboarding phase were followed. Through our manual analysis, we were able to identify potential vulnerabilities to ensure that the smart contract was secure and reliable.

5.2 Tools Used for Audit

In the course of our audit, we leveraged a suite of tools to bolster the security and performance of our program. While our team drew on their expertise and industry best practices, we also integrated various tools into our development environment. Noteworthy among them are Zetrix Development tool kit, Zetrix Ide. This holistic approach ensures a thorough analysis, uncovering potential issues that automated tools alone might overlook. Entersoft takes pride in utilizing these tools, which significantly contribute to the quality, security, and maintainability of our codebase.

6.0 Limitations on Disclosure and Use of this Report

This report contains information concerning potential details of the Zetrix Project and methods for exploiting them. Entersoft recommends that special precautions be taken to protect the confidentiality of both this document and the information contained herein. Security Assessment is an uncertain process, based on past experiences, currently available information, and known threats. All information security systems, which by their nature are dependent on human beings, are vulnerable to some degree. Therefore, while Entersoft considers the major security vulnerabilities of the analyzed systems to have been identified, there can be no assurance that any exercise of this nature will identify all possible vulnerabilities or propose exhaustive and operationally viable recommendations to mitigate those exposures. In addition, the analysis set forth herein is based on the technologies and known threats as of the date of this report. As technologies and risks change over time, the vulnerabilities associated with the operation of the Zetrix Javascript smart contract described in this report, as well as the actions necessary to reduce the exposure to such vulnerabilities will also change. Entersoft makes no undertaking to supplement or update this report based on changed circumstances or facts of which Entersoft becomes aware after the date hereof, absent a specific written agreement to perform the supplemental or updated analysis. This report may recommend that Entersoft use certain software or hardware products manufactured or maintained by other vendors. Entersoft bases these recommendations upon its prior experience with the capabilities of those products. Nonetheless, Entersoft does not and cannot warrant that a particular product will work as advertised by the vendor, nor that it will operate in the manner intended. This report was prepared by Entersoft for the exclusive benefit of Zetrix and is proprietary information. The Non-Disclosure Agreement (NDA) in effect between Entersoft and Zetrix governs the disclosure of this report to all other parties including product vendors and suppliers.