0.7

Synthetix Process Quality Review

Score 96%

This is a Synthetix Exchange Process Quality Audit completed on August 2020. It was performed using the Process Audit process (version 0.5) and is documented here. The audit was performed by ShinkaRex of Caliburn Consulting. Check out our Telegram.

The final score of the audit is 96%, a near perfect score. The breakdown of the scoring is in Scoring Appendix.

Summary of the Process

Very simply, the audit looks for the following declarations from the developer's site. With these declarations, it is reasonable to trust the smart contracts.

1. **Here is my smart contract on the blockchain**
2. **You can see it matches a software repository used to develop the code**
3. **Here is the documentation that explains what my smart contract does**
4. **Here are the tests I ran to verify my smart contract**
5. **Here are the audit(s) performed to review my code by third party experts**

Disclaimer

This report is for informational purposes only and does not constitute investment advice of any kind, nor does it constitute an offer to provide investment advisory or other services. Nothing in this report shall be considered a solicitation or offer to buy or sell any security, future, option or other financial instrument or to offer or provide any investment advice or service to any person in any jurisdiction. Nothing contained in this report constitutes investment advice or offers any opinion with respect to the suitability of any security, and the views expressed in this report should not be taken as advice to buy, sell or hold any security. The information in this report should not be relied upon for the purpose of investing. In preparing the information contained in this report, we have not taken into account the investment needs, objectives and financial circumstances of any particular investor. This information has no regard to the specific investment objectives, financial situation and particular needs of any specific recipient of this information and investments discussed may not be suitable for all investors.

Any views expressed in this report by us were prepared based upon the information available to us at the time such views were written. Changed or additional information could cause such views to change. All information is subject to possible correction. Information may quickly become unreliable for various reasons, including changes in market conditions or economic circumstances.

Executing Code Verification

This section looks at the code deployed on the Mainnet that gets audited and its corresponding software repository. The document explaining these questions is here. This audit will answer the questions;

1. Is the executing code address(s) readily available? (Y/N)
2. Is the code actively being used? (%)
3. Are the Contract(s) Verified/Verifiable? (Y/N)
4. Does the code match a tagged version in the code hosting platform? (%)
5. Is the software repository healthy? (%)

Is the executing code address(s) readily available? (Y/N)

> Answer: Yes

Actually finding the addresses was not immediately direct, but was simple enough. There is no developer page on the website. After clicking on the GitHub icon on the footer of the webpage and reading the "readme" file, I could link to the docs for synthetics. From their the addresses were clearly in the Developer Resource section. The page has all the details that could be desired.

They are available at Address 0x6AAeBDfbf23134eF8d81deB2E253f32394B2857B as indicated in the Appendix. This Audit only covers the contract FeePool. I chose fee pool in order to choose not the most obvious module. This contract was created on 30 June 2020.

Is the code actively being used? (%)

> Answer: 100%

Activity on Fee Pool seems to ebb and flow with some days 0 transaction, others more than 50.

Activity is average 13 transactions a day, as indicated in the Appendix.

Percentage Score Guidance

100%	More than 10 transactions a day
70%	More than 10 transactions a week
40%	More than 10 transactions a month
10%	Less than 10 transactions a month
0%	No activity

Are the Contract(s) Verified/Verifiable? (Y/N)

> Answer: Yes

0x5eF0de4bd373e435341Cd82311dfb13d5E8fdEf5 is the Etherscan verified contract address.

Does the code match a tagged version on a code hosting platform? (%)

> Answer: 100%

The releases are either biased in the documentation section. There are frequent releases. The file matched perfectly.

Guidance:

100%	Code matches and Repository was clearly labelled
60 %	Code matches but no labelled repository. Repository was found manually
30%	Code does match perfectly and repository was found manually
0%	Matching Code could not be found

GitHub address : https://github.com/Synthetixio/synthetix

Deployed contracts in the following file;

FeePool.rar 17KB
Binary

Matching Repository: https://github.com/Synthetixio/synthetix/releases/tag/v2.25.0-alpha

Is development software repository healthy? (%)

> Answer: 100%

Clearly this repository was used for the majority of the development. It is still in use today. It has nearly 3000 commit and 3000 branches.

Documentation

This section looks at the software documentation. The document explaining these questions is here.

Required questions are;

1. Is there a whitepaper? (Y/N)
2. Are the basic application requirements documented? (Y/N)
3. Do the requirements fully (100%) cover the deployed contracts? (%)
4. Are there sufficiently detailed comments for all functions within the deployed contract code (%)
5. Is it possible to trace software requirements to the implementation in code (%)

Is there a whitepaper? (Y/N)

> ✓ Answer: Yes

Location: https://docs.synthetix.io/litepaper/

Are the basic application requirements documented? (Y/N)

> ✓ Answer: Yes

Location: https://docs.synthetix.io/

How to improve this score

Write the document based on the deployed code. For guidance, refer to the SecurEth System Description Document.

Do the requirements fully (100%) cover the deployed contracts? (%)

> ✓ Answer: 100%

The documentation for Synthetix is exceptional. The description for the FeePool module includes diagrams for an inheritance graph and related contracts, descriptions of all constants and variables and detailed descriptions of each function with the source for the function identified. Clearly this is the new baseline on which other applications will be judged on documentation.

Are there sufficiently detailed comments for all functions within the deployed contract code (%)

> ⓘ Answer: 65%

The first impression is to give a very high score for commenting because comments are clear, detailed and

included where they add value. The percentage of code to commenting is actually low compared to many other applications at 31%. Numbers in excess of 70% have been seen on well-documented other applications. Part of the reason may be that there is just so much code included. At 9,000 lines over 45 modules this is one of the largest applications audited to date. Based on the commenting the code percentage score of about 45% would be in line with other audits. Given the quality I will increase it to 65%.

Code examples are in the Appendix. As per the SLOC, there is 31% commenting to code.

How to improve this score

This score can improve by adding comments to the deployed code such that it comprehensively covers the code. For guidance, refer to the SecurEth Software Requirements.

Is it possible to trace requirements to the implementation in code (%)

> ✓ Answer: 100%

Each function is clearly documented. Next of the documentation is a link directly to the source code. This is exactly the type of traceability we are looking for.

Guidance:
100% - Clear explicit traceability between code and documentation at a requirement level for all code
60% - Clear association between code and documents via non explicit traceability
40% - Documentation lists all the functions and describes their functions
0% - No connection between documentation and code

How to improve this score

This score can improve by adding traceability from requirements to code such that it is clear where each requirement is coded. For reference, check the SecurEth guidelines on traceability.

Testing

This section looks at the software testing available. It is explained in this document. This section answers the following questions;

1. Full test suite (Covers all the deployed code) (%)
2. Code coverage (Covers all the deployed lines of code, or explains misses) (%)
3. Scripts and instructions to run the tests (Y/N)
4. Packaged with the deployed code (Y/N)
5. Report of the results (%)
6. Formal Verification test done (%)
7. Stress Testing environment (%)

Is there a Full test suite? (%)

> Answer: 100%

Clearly there is a full test suite. Test line to code ratio is 375%

Code coverage (Covers all the deployed lines of code, or explains misses) (%)

> Answer: 99%

Code coverage for the deployed release is 99% as per GitHub CodeCov, see the Appendix.

Guidance:
100% - Documented full coverage
99-51% - Value of test coverage from documented results
50% - No indication of code coverage but clearly there is a reasonably complete set of tests
30% - Some tests evident but not complete
0% - No test for coverage seen

How to improve this score

This score can improve by adding tests achieving full code coverage. A clear report and scripts in the software repository will guarantee a high score.

Scripts and instructions to run the tests (Y/N)

> Answer: Yes

Sythetix has a full document on their automated test procedures.

Packaged with the deployed code (Y/N)

> Answer: Yes

Report of the results (%)

> Answer: 80%

The automated code coverage report answers most of the questions. What is missing is a stand alone report indicating the pass of all tests and describing the few misses. But clearly, most questions are answered.

How to improve this score

Add a report with the results. The test scripts should generate the report or elements of it.

Formal Verification test done (%)

> ⚠ Answer: 0%

No evidence of Formal Verification was found.

Stress Testing environment (%)

> ✓ Answer: 100%

The address page includes test addresses for Ropsten, Rinkeby and Kovan. The Ropsten address' at a minimum are still in regular use.

Audits

> ✓ Answer: 100%

The audit history page shows regular d audits taking place. There were eight audits in 2020 alone (before mid-August) from two independent auditors. The audit results and documentation appears meticulous and complete.

Guidance:

1. Multiple Audits performed before deployment and results public and implemented or not required (100%)
2. Single audit performed before deployment and results public and implemented or not required (90%)
3. Audit(s) performed after deployment and no changes required. Audit report is public. (70%)
4. No audit performed (20%)
5. Audit Performed after deployment, existence is public, report is not public and no improvements deployed (0%)

Appendices

Author Details

The author of this audit is Rex of Caliburn Consulting.

Email : rex@caliburnc.com Twitter : @ShinkaRex

I started with Ethereum just before the DAO and that was a wonderful education. It showed the importance of code quality. The second Parity hack also showed the importance of good process. Here my aviation background offers some value. Aerospace knows how to make reliable code using quality processes.

I was coaxed to go to EthDenver 2017 and there I started SecuEth.org with Bryant and Roman. We created guidelines on good processes for blockchain code development. We got EthFoundation funding to assist in their development.

Process Quality Audits are an extension of the SecurEth guidelines that will further increase the quality processes in Solidity and Vyper development.

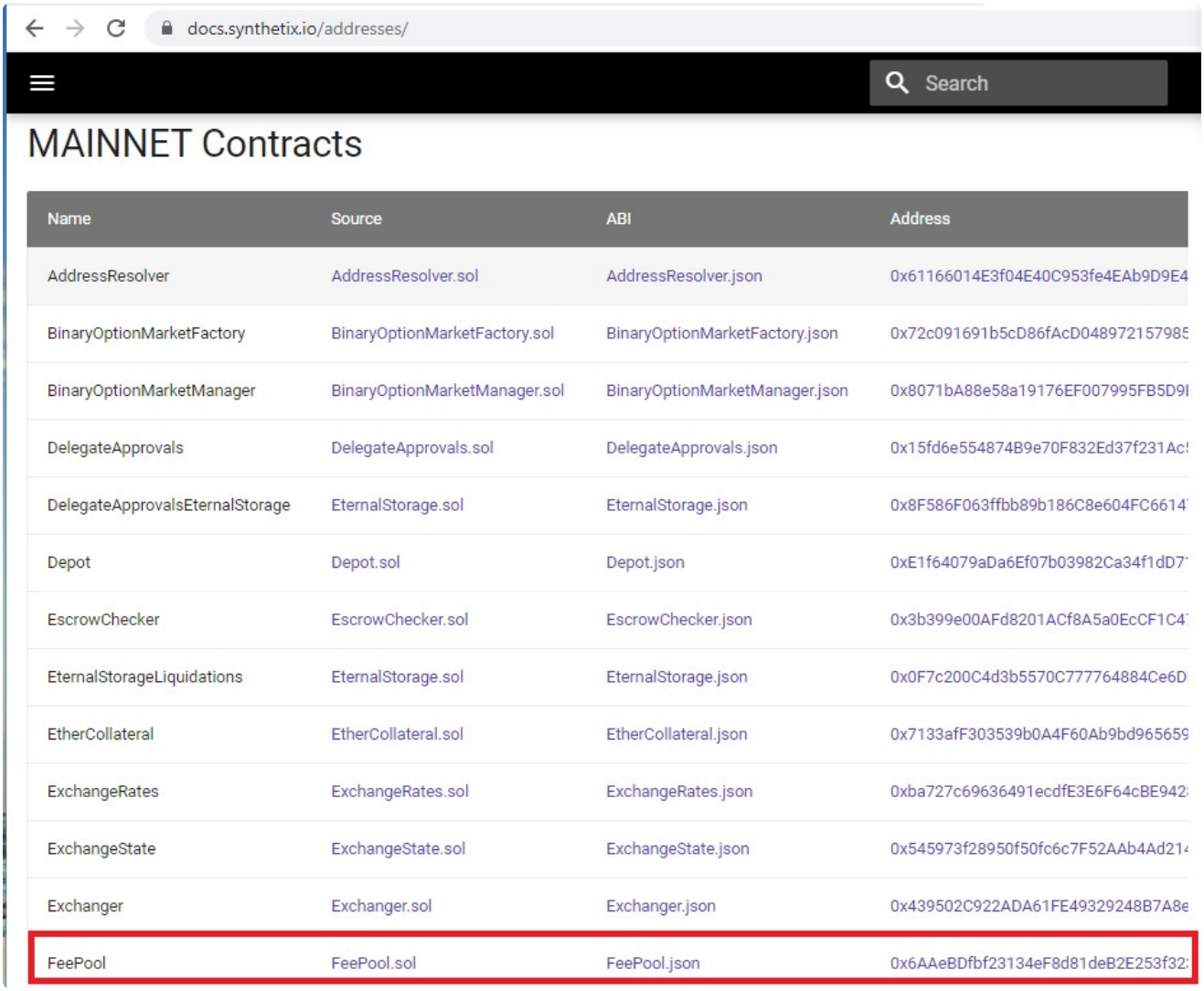
Career wise I am a business development for an avionics supplier.

Scoring Appendix

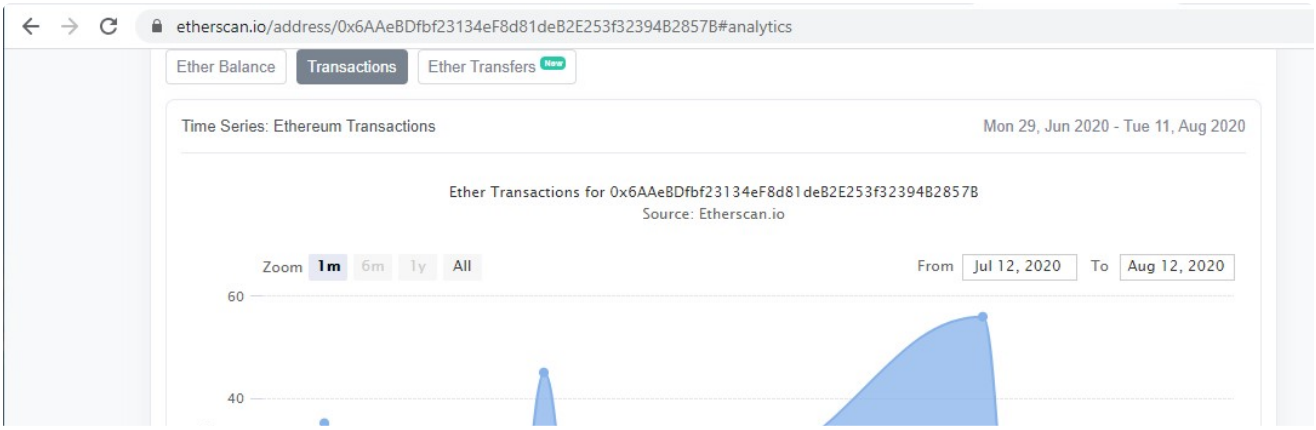
	Total	Synthetix	
PQ Audit Scoring Matrix (v0.4 and 0.5)	**Points**	**Answer**	**Points**
Total	**240**		229.45
Executing Code Verification			**96%**
1. Is the executing code address(s) readily available? (Y/N)	**30**	y	30
2. Is the code actively being used? (%)	**5**	100%	5
3. Are the Contract(s) Verified/Verifiable? (Y/N)	**5**	Y	5
4. Does the code match a tagged version on a code hosting platform? (%)	**20**	100%	20
5. Is development software repository healthy? (%)	**10**	100%	10
Code Documentation			
1. Is there a whitepaper? (Y/N)	**5**	y	5
2. Are the basic application requirements documented? (Y/N)	**10**	y	10
3. Do the requirements fully (100%) cover the deployed contracts? (%)	**15**	100%	15
4. Are there sufficiently detailed comments for all functions within the deployed contract code (%)	**10**	65%	6.5
5. Is it possible to trace requirements to the implementation in code (%)	**5**	100%	5
Testing			
1. Full test suite (Covers all the deployed code) (%)	**20**	100%	20
2. Code coverage (Covers all the deployed lines of code, or explains misses) (%)	**5**	99%	4.95
3. Scripts and instructions to run the tests? (Y/N)	**5**	Y	5
4. Packaged with the deployed code (Y/N)	**5**	Y	5
5. Report of the results (%)	**10**	80%	8
6. Formal Verification test done (%)	**5**	0%	0
7. Stress Testing environment (%)	**5**	100%	5
Audits			
Audit done	**70**	100%	70
Section Scoring			
Executing Code Verification	70	100%	

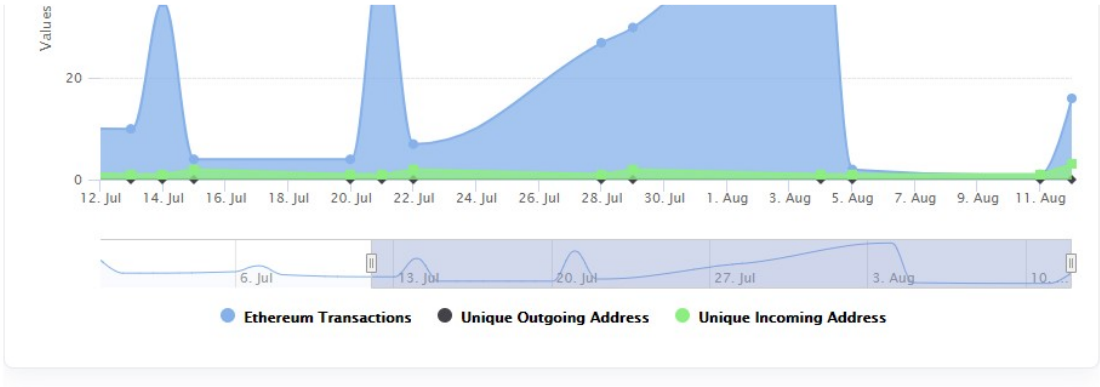
Documentation	45	92%	
Testing	55	87%	
Audits	70	100%	

Executing Code Appendix



Code Used Appendix





Example Code Appendix

```
// https://docs.synthetix.io/contracts/FeePool
contract FeePool is Owned, Proxyable, SelfDestructible, LimitedSetup, MixinResolver, IFeeP
    using SafeMath for uint;
    using SafeDecimalMath for uint;

    // Exchange fee may not exceed 10%.
    uint public constant MAX_EXCHANGE_FEE_RATE = 1e18 / 10;

    // Where fees are pooled in sUSD.
    address public constant FEE_ADDRESS = 0xfeEFEEfeefEeFeefEEFEEfEeFeefEEFeeFEEFEeF;

    // sUSD currencyKey. Fees stored and paid in sUSD
    bytes32 private sUSD = "sUSD";

    // This struct represents the issuance activity that's happened in a fee period.
    struct FeePeriod {
        uint64 feePeriodId;
        uint64 startingDebtIndex;
        uint64 startTime;
        uint feesToDistribute;
        uint feesClaimed;
        uint rewardsToDistribute;
        uint rewardsClaimed;
    }

    // A staker(mintr) can claim from the previous fee period (7 days) only.
    // Fee Periods stored and managed from [0], such that [0] is always
    // the current active fee period which is not claimable until the
    // public function closeCurrentFeePeriod() is called closing the
    // current weeks collected fees. [1] is last weeks feeperiod
    uint8 public constant FEE_PERIOD_LENGTH = 2;

    FeePeriod[FEE_PERIOD_LENGTH] private _recentFeePeriods;
    uint256 private _currentFeePeriod;

    // How long a fee period lasts at a minimum. It is required for
    // anyone to roll over the periods, so they are not guaranteed
    // to roll over at exactly this duration, but the contract enforces
```

```
    // that they cannot roll over any quicker than this duration.
    uint public feePeriodDuration = 1 weeks;
    // The fee period must be between 1 day and 60 days.
    uint public constant MIN_FEE_PERIOD_DURATION = 1 days;
    uint public constant MAX_FEE_PERIOD_DURATION = 60 days;

    // Users are unable to claim fees if their collateralisation ratio drifts out of targe
    uint public targetThreshold = (1 * SafeDecimalMath.unit()) / 100;

       /* ========== MUTATIVE FUNCTIONS ========== */

    /**
     * @notice Logs an accounts issuance data per fee period
     * @param account Message.Senders account address
     * @param debtRatio Debt percentage this account has locked after minting or burning th
     * @param debtEntryIndex The index in the global debt ledger. synthetixState.issuanceDa
     * @dev onlyIssuer to call me on synthetix.issue() & synthetix.burn() calls to store th
     * per fee period so we know to allocate the correct proportions of fees and rewards pe
     */
    function appendAccountIssuanceRecord(
        address account,
        uint debtRatio,
        uint debtEntryIndex
    ) external onlyIssuer {
        feePoolState().appendAccountIssuanceRecord(
            account,
            debtRatio,
            debtEntryIndex,
            _recentFeePeriodsStorage(0).startingDebtIndex
        );

        emitIssuanceDebtRatioEntry(account, debtRatio, debtEntryIndex, _recentFeePeriodsSto
    }

    /**
     * @notice Set the fee period duration
     */
    function setFeePeriodDuration(uint _feePeriodDuration) external optionalProxy_onlyOwne
        require(_feePeriodDuration >= MIN_FEE_PERIOD_DURATION, "value < MIN_FEE_PERIOD_DURA
        require(_feePeriodDuration <= MAX_FEE_PERIOD_DURATION, "value > MAX_FEE_PERIOD_DURA

        feePeriodDuration = _feePeriodDuration;

        emitFeePeriodDurationUpdated(_feePeriodDuration);
    }


```

SLOC Appendix

Solidity Contracts

Language	Files	Lines	Blanks	Comments	Code	Complex
Solidity	45	9218	1771	1764	5683	546

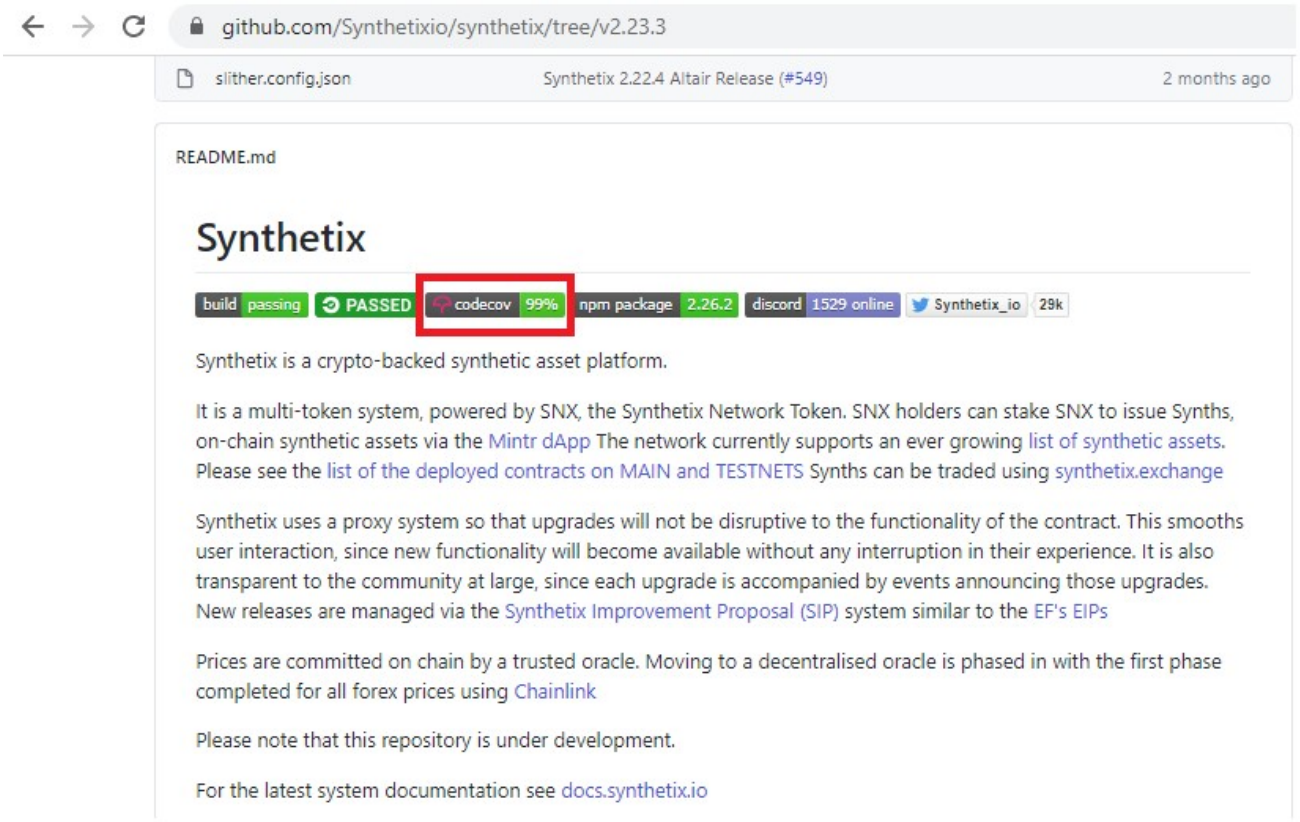
Comments to Code 1764 / 5683 = 31%

Javascript Tests

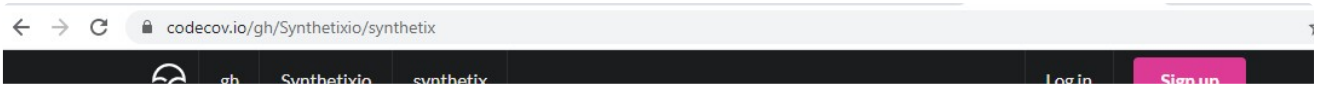
Language	Files	Lines	Blanks	Comments	Code	Complex
JavaScript	42	26,743	3534	1855	21,354	477

Tests to Code 21,354/ 5683 = 375%

Code Coverage Appendix

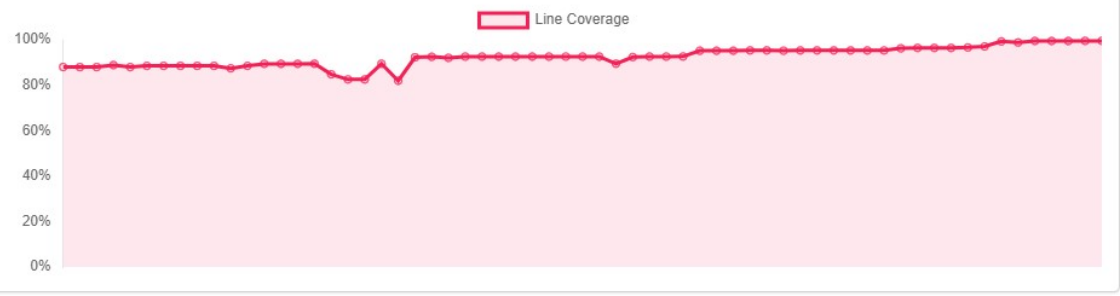


Code Coverage Report

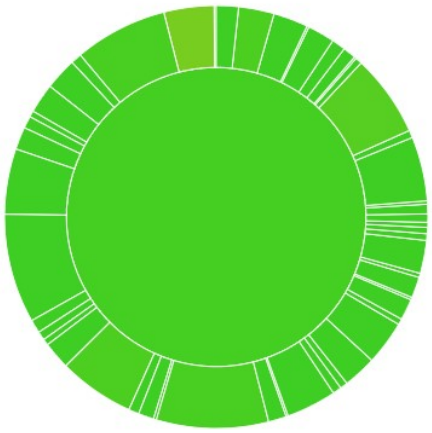


Showing min per day coverage for the last 6 months

COVERAGE CHART



COVERAGE SUNBURST



ALL RECENT COMMITS

Update deps. Browse Report
zyzek 13 hours ago #559 1fb8450 CI Passed

Fixing broken tests Browse Report
justinjmoses a day ago #658 73ed8a2 CI Passed

Minor optimizatoin Browse Report
justinjmoses 2 days ago #658 e2f03a3 CI Passed

Upgrading to use chainlink 0.0.9 interfaces Browse Report
justinjmoses 2 days ago #658 1469ebb CI Passed

Merge branch 'develop' into sip-76-warning-flags-invalid-rates Browse Report
justinjmoses 2 days ago #658 8964359 CI Passed

Webpack support for a browser bundle (#657) Browse Report
justinjmoses
2 days ago add-precommit-format b4e2d12 CI Failed