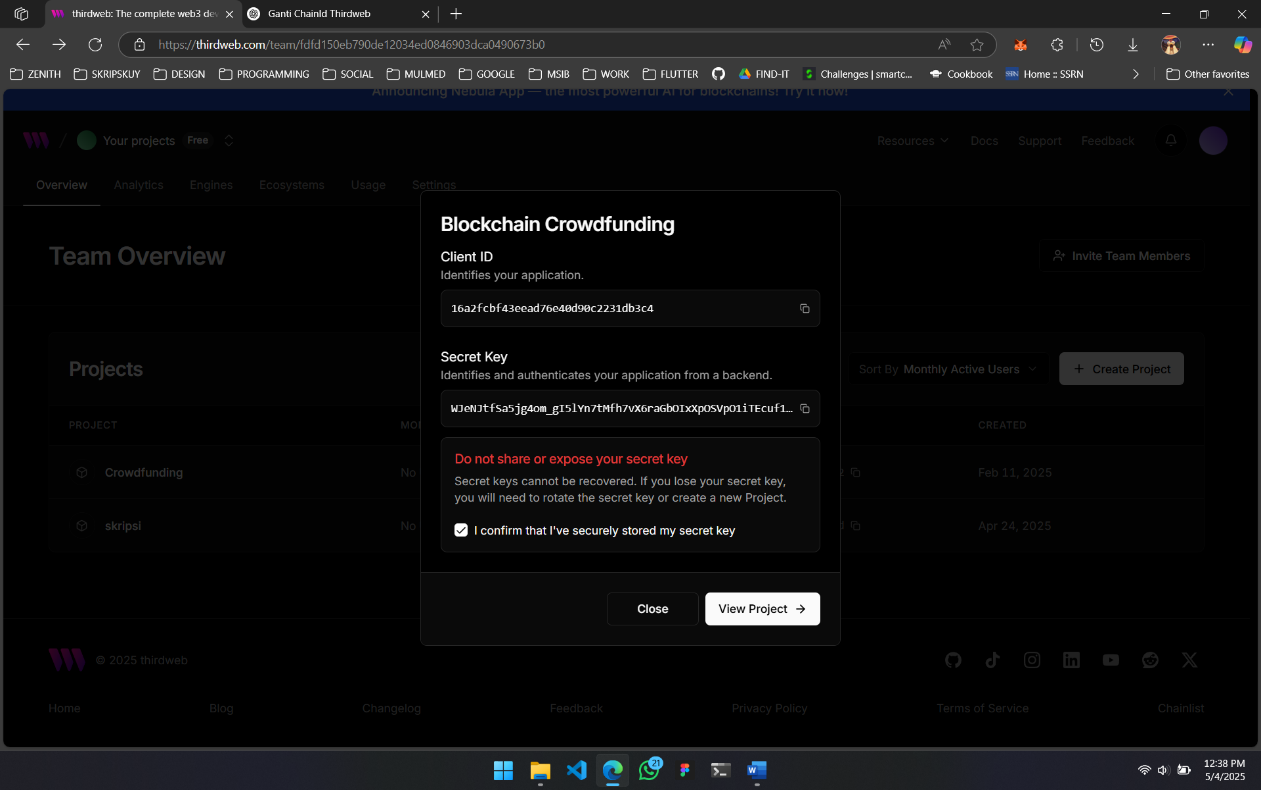
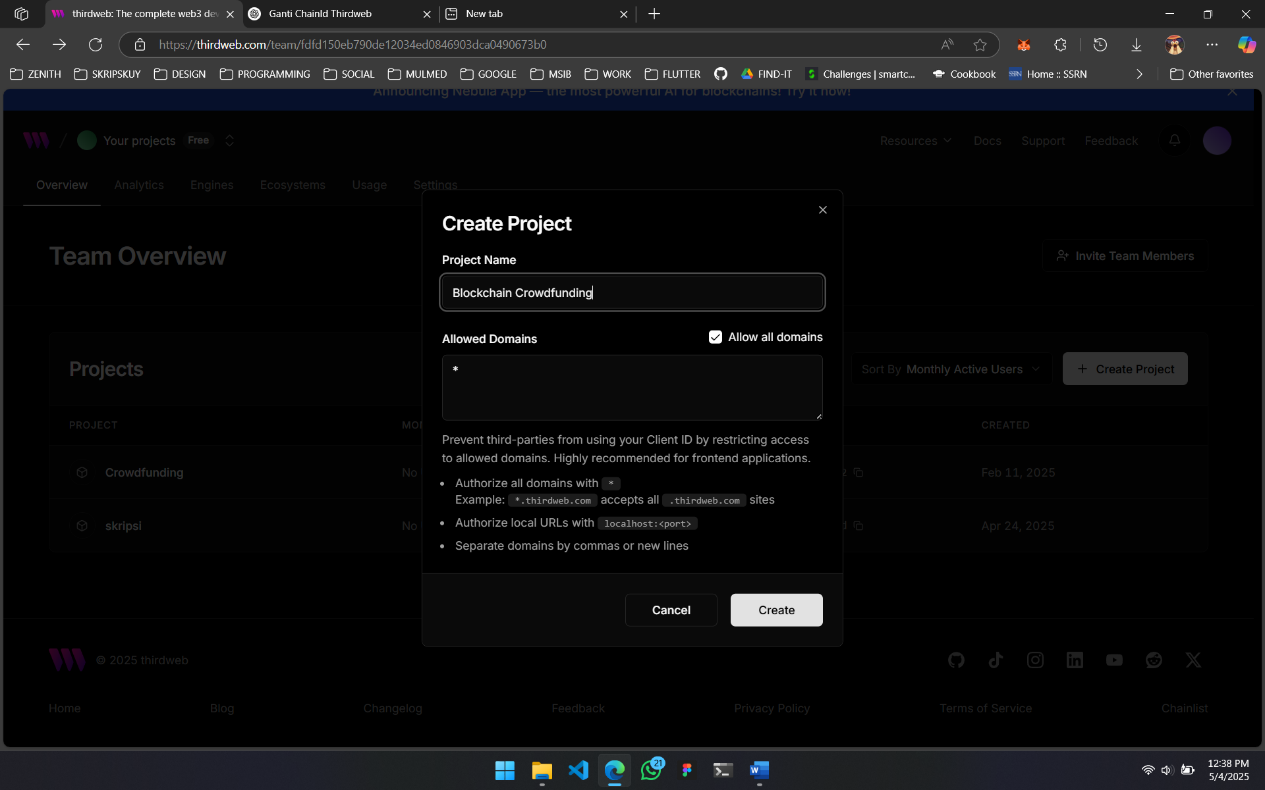
**BAB IV HASIL DAN PEMBAHASAN**

**4.1 Implementasi Smart Contract**

Berdasarkan model smart contract yang telah dibuat maka didapatkan hasil beberapa kontrak meliputi kontrak crowdfunding, kontrak staking dan reward token, kontrak token drop dan kontrak staking lengkap dengan method dan atribut masig-masing kontrak. Hasil dari implementasi masing-masing kontrak terdapat pada lampiran. Setelah masing-masing kontrak telah diimplementasi, tahap selanjutnya adalah melakukan deploy pada EVM menggunakan Thirdweb, Thirdweb digunakan sebagai EVM untuk membantu integrasi anatara smart contract dengan aplikasi. Untuk mengintegrasikan sistem maka dilakukan langkah-langkah berikut:

1. Membuat proyek di Thirdweb



**Gambar 4.x** Create project pada EVM

1. Deploy smart contract



**Gambar 4.x** Deploy smart contract ke EVM

Ketika proses deploy berhasil, sistem akan menampilkan tautan url pada terminal. Tautan tersebut digunakan untuk melakukan konfigurasi masing-masing kontrak untuk memilih chain atau jaringan blockchain dimana kontrak akan di deploy.

**4.2 Implementasi Web3**

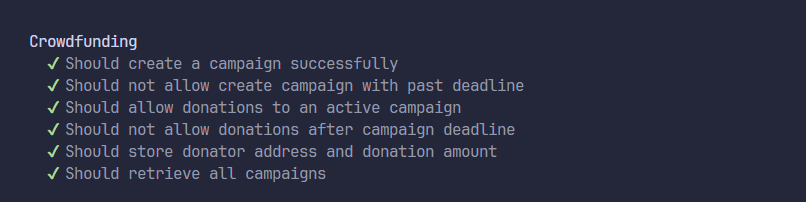
Kontrak yang sudah dideploy pada jaringan Sepolia Ethereum lewat Ethereum Virtual Machine menggunaka thirdweb akan dilakukan integrasi dengan aplikasi. Aplikasi dikembangkan menggunakan bahasa Javascript dengan menggunakan library React JS. Interface aplikasi terdiri dari beberapa halaman, antaralain; halaman dashboard; halaman detail; halaman history; halaman claim token dan halaman staking. Hasil interface web3 pada lampiran.

**4.3 Uji Fungsionalitas**

Setelah dilakukan implementasi smart contract akan di uji dari segi fungsionalitasnya, pengujian bermaksud untuk memastikan bahwa semua fungsi yang dituliskan pada smart contract berjalan dengan baik. Pengujian dilakukan pada masing-masing kontrak menggunakan Hardhat JS. Berikut ini adalah test case untuk smart contract Crowdfunding:

1. Kontrak dapat membuat kampanye
2. Kontrak dapat melarang kampanye dengan deadline masa lalu
3. Kontrak dapat melakukan donasi ke kampanye yang aktif
4. Kontrak dapat melarang donasi setelah lewat deadline
5. Kontrak dapat menyimpan data donator dan jumlah donasi
6. Kontrak dapat melihat data seluruh kampanye

Kontrak akan diuji mengunakan Hardhat JS dengan menginputkan perintah pada terminal npx hardhat test ./test/01\_Cowdfunding.test.js. Kode untuk melakukan testing terdapat pada lampiran () dan di dapatkan hasil testing seperti gambar dibawah ini.

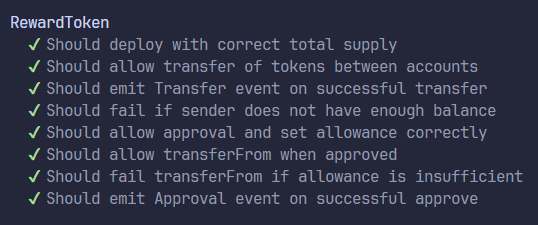
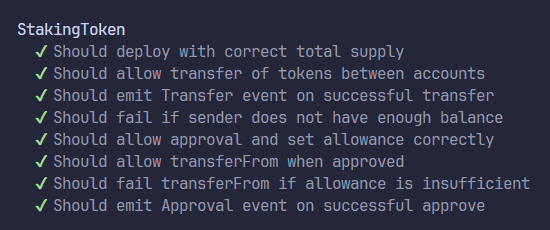


**Gambar 4.x** Hasil pengujian kontrak crowdfunding dengan Hardhat

Dari proses testing fungsional menggunakan Hardhat JS, didapatkan hasil dari testing smart contract crowdfunding yang berhasil memenuhi test case. Selanjutnya, testing fungsional akan dilakukan pada kontrak staking dan reward token dengan test case sebagai berikut:

1. Kontrak dapat mendeploy token dengan suplai tertentu
2. Kontrak dapat Mentransfer token antar akun
3. Kontrak dapat Mencatat transfer yang berhasil
4. Kontrak dapat Membatalkan transfer jika tidak cukup token
5. Kontrak dapat Menyetujui akun lain untuk transfer
6. Kontrak dapat Mentransfer token sesudah di setujui
7. Kontrak dapat Membatalkan transfer akun yang disetujui jika token kurang
8. Kontrak dapat Mencatat persetujuan yang berhasil

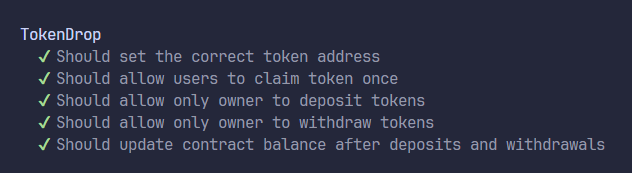
Kontrak akan diuji mengunakan Hardhat JS dengan menginputkan perintah pada terminal npx hardhat test ./test/02\_StakingToken.test.js ./test/03\_RewardToken.test.js. Kode untuk melakukan testing terdapat pada lampiran () dan di dapatkan hasil testing seperti gambar dibawah ini.

**Gambar 4.x** Hasil pengujian kontrak staking dan reward token dengan Hardhat

Dari proses testing fungsional menggunakan Hardhat JS, didapatkan hasil dari testing smart contract staking dan reward token yang berhasil dan memenuhi test case. Selanjutnya, testing fungsional akan dilakukan pada kontrak token drop dengan test case sebagai berikut:

1. Menetapkan alamat reward token
2. Mengizinkan user klaim token
3. Mengizinkan hanya owner smart contract saja untuk deposit token
4. Mengizinkan hanya owner smart contract saja untuk menarik token
5. Memperbarui suplai kontrak setelah deposit dan withdraw token

Kontrak akan diuji mengunakan Hardhat JS dengan menginputkan perintah pada terminal npx hardhat test ./test/04\_TokenDrop.test.js. Kode untuk melakukan testing terdapat pada lampiran () dan di dapatkan hasil testing seperti gambar dibawah ini.

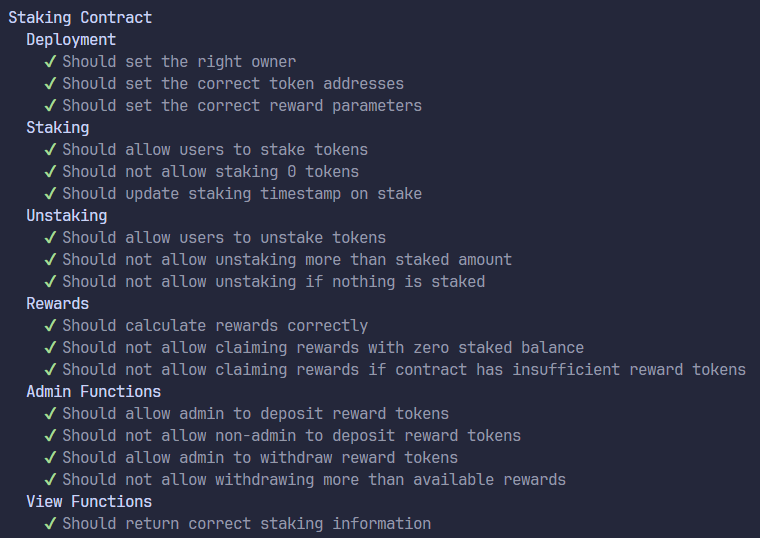


**Gambar 4.x** Hasil pengujian kontrak token drop dengan Hardhat

Dari proses testing fungsional menggunakan Hardhat JS, didapatkan hasil dari testing smart contract token drop yang berhasil dan memenuhi test case. Selanjutnya, testing fungsional akan dilakukan pada kontrak staking dengan test case sebagai berikut:

1. Menetapkan alamat owner, alamat token staking, dan alamat token reward
2. Mengizinkan user untuk staking, memastikan jumlah token staking adalah 1, dan melakukan update timestamp
3. Mengizinkan user untuk unstaking, memastikan jumlah token yang di-unstaking tidak lebih dari 1 dan melarang user untuk unstaking jika tidak melakukan staking
4. Mengkalkulasi jumlah reward, mengizinkan hanya owner yang bisa menarik token reward, dan melaran user menarik reward lebih dari jumlah kalkulasi
5. Menampilkan data staking

Kontrak akan diuji mengunakan Hardhat JS dengan menginputkan perintah pada terminal npx hardhat test ./test/05\_Staking.test.js. Kode untuk melakukan testing terdapat pada lampiran () dan di dapatkan hasil testing seperti gambar dibawah ini.



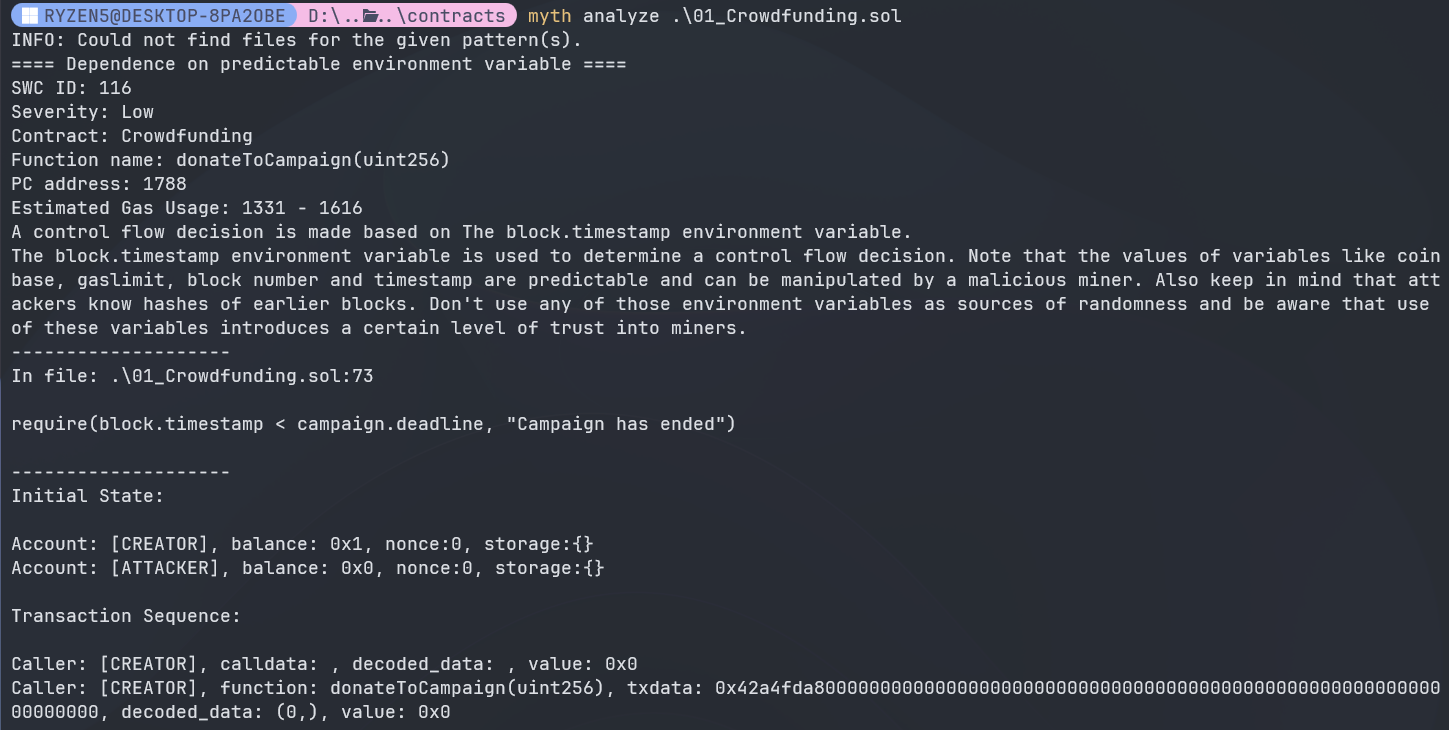
**Gambar 4.x** Hasil pengujian kontrak staking dengan Hardhat

Dari proses testing fungsional menggunakan Hardhat JS, didapatkan hasil dari testing smart contract token drop yang berhasil dan memenuhi test case. Dari masing-masing kontrak yang sudah dilakukan testing secara fungsional mendapatkan hasil bahwa smart contract yang dibangun dapat beroperasi sesuai dengan fungsi yang sudah dituliskan dalam smart contract terkait. Hal ini dapat dilihat dari hasil testing menggunakan Hardhat dan hasil yang ditampilkan Hardhat JS memenuhi semua test case yang ada.

**4.4 Uji Non-fungsionalitas**

Pengujian non-fungsionalitas dilakukan untuk mendeteksi potensi celah keamanan smart contract, memastikan integritas data dan availability data. Mythril digunakan sebagai tools untuk mendeteksi potensi celah keamanan. Sedangkan sepoliaetherscan digunakan untuk memastikan integritas dan availability data yang disimpan pada jaringan blockchain.

Hasil pengujian menggunakan Mythril pada kontrak Crowdfunding adalah sebagai berikut:



**Gambar 4.x** Hasil pengujian Mythril pada kontrak crowdfunding

Pada hasil testing kontrak Crowdfunding ditemukan tingkat keparahan kode yang rendah, masalah ditemukan pada fungsi donateToCampaign(uint256). Berdasarkan pada deskripsi program, fungsi tersebut menggunakan block.timestamp untuk menentukan apakah kampanye masih terbuka atau sudah tutup. Nilai block.timestamp adalah nilai yang dapat diprediksi dan dapat dimanipulasi oleh miner dengan memajukan timestap beberapa detik. Hal ini akan memperngaruhi logika kontrak seperti membuat donasi masih bisa dilakukan meskipun sudah melewati batas waktu.

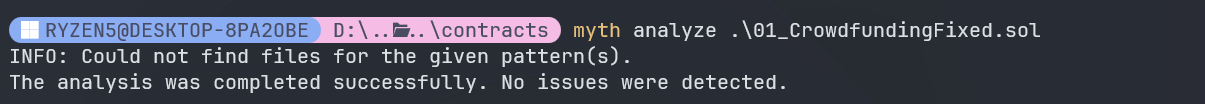
Untuk menangani kerentanan pada kode, dilakukan perbaikan kode pada smart contract pada beberapa bagian dengan menambahkan validasi pada bagian fungsi donateToCampaign. Perbaikan pada smart contract terletak pada kode dibawah.

|  |
| --- |
| ```  event *CampaignStatusChanged*(uint256 indexed *campaignId*, bool *isActive*);  function *donateToCampaign*(uint256 *\_id*) public payable {          uint256 amount = msg.value;          Campaign storage campaign = campaigns[\_id];          require(campaign.isActive, "Campaign is not active");            if (block.timestamp >= campaign.deadline) {              campaign.isActive = false;              emit *CampaignStatusChanged*(\_id, false);              revert("Campaign has ended");          }            require(amount > 0, "Donation must be greater than zero");          campaign.donators.*push*(msg.sender);          campaign.donations.*push*(amount);            campaign.amountCollected += amount;            emit *DonationReceived*(\_id, msg.sender, amount);          (bool sent,) = payable(campaign.owner).call{value: amount}("");          require(sent, "Failed to send donation to campaign owner");      }  ``` |

**Kode 6.** Perbaikan smart contract crowdfunding

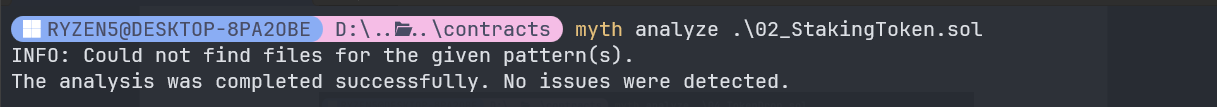
Pada fungsi donateToCampaign ditambahkan validasi yang lebih aman untuk memvalidasi deadline dari kampanye. Validasi tidak langsung menggunakan block timestamp, namun menggunakan variable isActive dan menambahkan event CampaignStatusChanged untuk mencatat perubahan pada nilai variabel isActive.

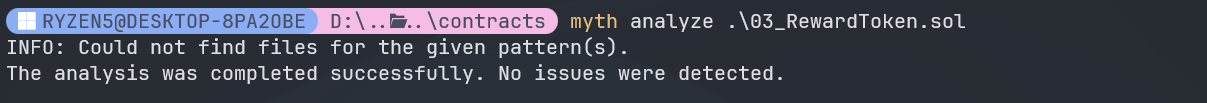
Setelah ditambahkan kode untuk validasi pada fungsi donateToCampaign, selanjutnya akan dilakukan Analisa ulang menggunakan Mythril. Testing dilakukan untuk memastikan apakah kode smart contract yang sudah di perbaiki masih memiliki celah keamanan atau tidak. Hasil pengujian ulang smart contract Crowdfunding menggunakan Mythril adalah sebagai berikut:



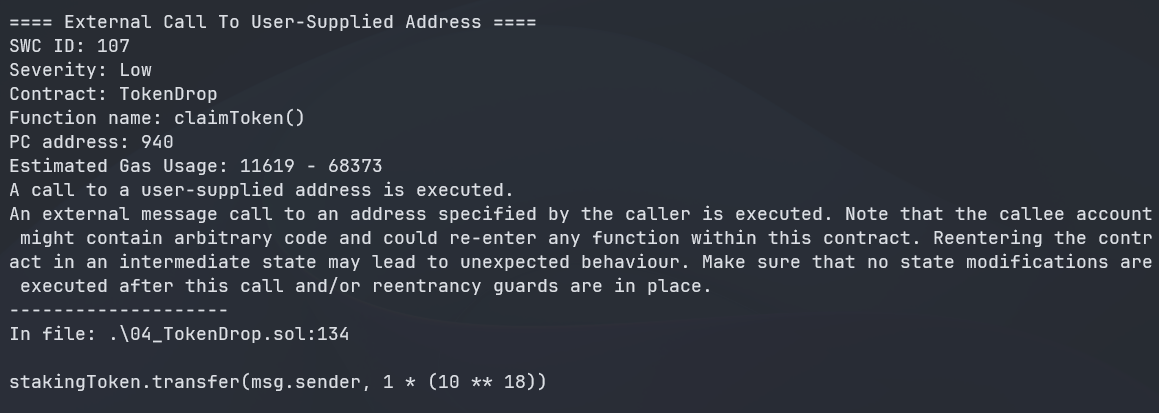
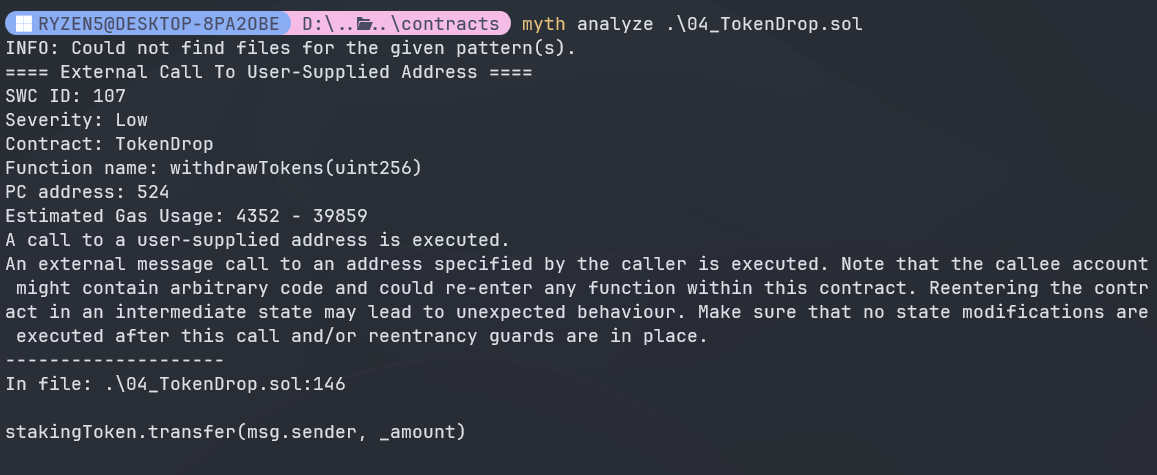
**Gambar 4.x** Hasil pengujian kembali smart contract crowdfuding

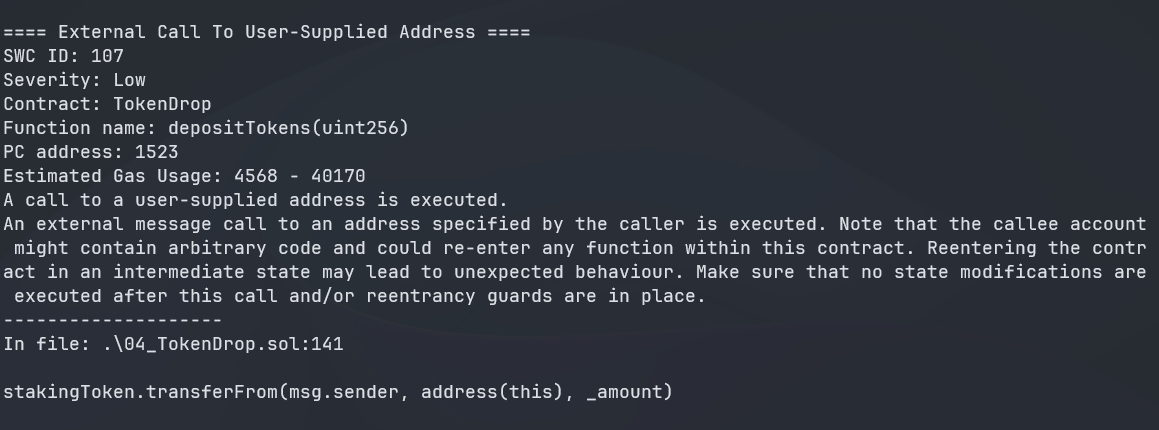
Hasil pengujian menggunakan Mythril pada kontrak Staking Token dan RewardToken adalah sebagai berikut:



**Gambar 4.x** Hasil pengujian Mythril pada kontrak Staking Token dan Reward Token

Pada hasil testing kontrak Staking Token dan Reward Token tidak ditemukan keparahan atau kerentanan kode pada proses analisa menggunakan Mythril. Kemudian pada hasil pengujian menggunakan Mythril pada kontrak Token Drop adalah sebagai berikut:





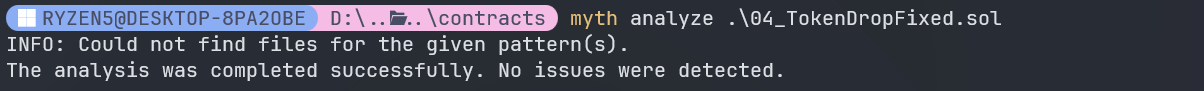
**Gambar 4.x** Hasil pengujian Mythril pada kontrak token drop

Hasil pengujian smart contract Token Drop menampilkan beberapa kerentanan atau keparahan kode. Kerentanan terdapat pada tiga fungsi yakni fungsi withdrawToken, fungsi claimToken dan fungsi depositTokens. Pada ketiga fungsi terdapat ancaman yang sama, berdasarkan pada deskripsi program, ketiga fungsi tersebut rentan terhadap potensi celah keamanan yakni *Reentrancy Attack. Reentrancy Attack* adalah sebuah mekanisme (cari jurnal aja lek ku). Untuk mengatasi kerentanan ini, smart contract perlu diperbaiki pada ketiga fungsi tersebut. Hasil perbaikan dari smart contract TokenDrop adalah sebagai berikut:

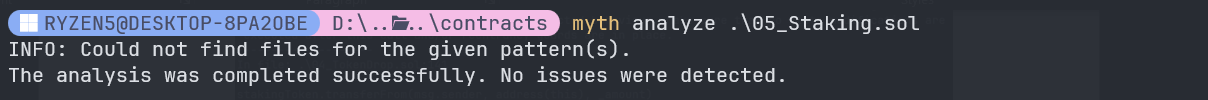
|  |
| --- |
| ```  modifier *nonReentrant*() {          require(!\_locked, "ReentrancyGuard: reentrant call");          \_locked = true;          \_;          \_locked = false;      }      function *claimToken*() external *nonReentrant* {          require(!hasClaimed[msg.sender], "You have already claimed");          hasClaimed[msg.sender] = true;          uint256 claimAmount = 1 \* (10 \*\* 18);            require(stakingToken.*balanceOf*(address(this)) >= claimAmount, "Insufficient balance");            emit *Claimed*(msg.sender, claimAmount);          bool success = stakingToken.*transfer*(msg.sender, claimAmount);          require(success, "Token transfer failed");      }      function *depositTokens*(uint256 *\_amount*) external *onlyOwner* *nonReentrant* {          require(\_amount > 0, "Amount must be greater than zero");            uint256 balanceBefore = stakingToken.*balanceOf*(address(this));            bool success = stakingToken.*transferFrom*(msg.sender, address(this), \_amount);          require(success, "Token transfer failed");            uint256 balanceAfter = stakingToken.*balanceOf*(address(this));          require(balanceAfter >= balanceBefore + \_amount, "Token deposit verification failed");            emit *TokensDeposited*(msg.sender, \_amount);      }      function *withdrawTokens*(uint256 *\_amount*) external *onlyOwner* *nonReentrant* {          require(\_amount > 0, "Amount must be greater than zero");          require(stakingToken.*balanceOf*(address(this)) >= \_amount, "Insufficient balance");            emit *TokensWithdrawn*(msg.sender, \_amount);            bool success = stakingToken.*transfer*(msg.sender, \_amount);          require(success, "Token transfer failed");      }  ``` |

**Kode.7** Perbaikan smart contract token drop

Perbaikan fungsi claimToken, deposiToken dan withdrawToken dilakukan dengan menambahkan modifier nonreentrant untuk mencegah pemanggilan fungsi kembali, selain itu pada masing-masing kontrak ditambahkan validasi untuk memastikan keamanan dan tidak melakukan eksekusi fungsi secara langsung. Setelah dilakukan perbaikan maka smart contract akan diuji kembali. Hasil pengujian kembali smart contract TokenDrop sudah tidak lagi ditemukan keparahan atau kerentanan yang ditampilkan pada gambar dibawah.

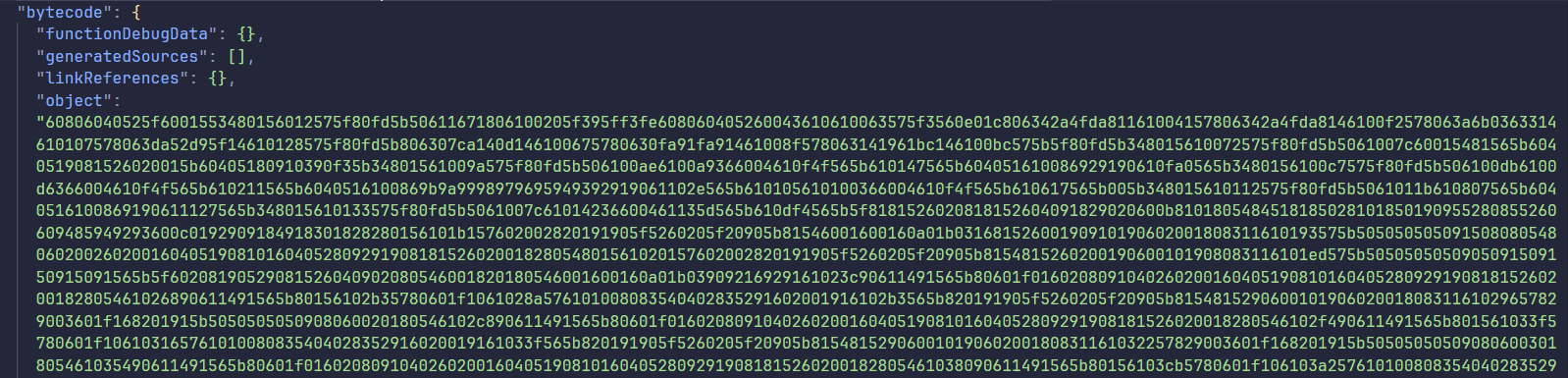
 **Gambar 4.x** Hasil pengujian kembali kontrak token drop

Hasil pengujian menggunakan Mythril pada kontrak Staking adalah sebagai berikut:

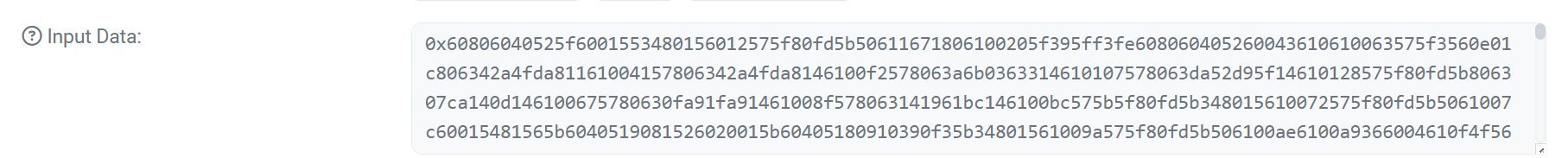


**Gambar 4.x** Hasil pengujian menggunakan Mythril pada kontrak Staking

Pengujian juga dilakukan secara onchain dengan melakukan tracking pada jaringan blockchain secara langsung pada exploler. Proses pengujian untuk memastikan integritas data dilakukan dengan membandingkan bytecode dari smart contract yang dikompilasi pada computer lokal dengan bytecode yang disimpan pada jaringan blockchain. Bytecode didapat pada file berbentuk json yang diciptakan ketika melakukan kompilasi kontrak. Berikut adalah bytecode hasil kompilasi kontrak pada computer lokal dan bytecode yang disimpan pada jaringan blockchain.

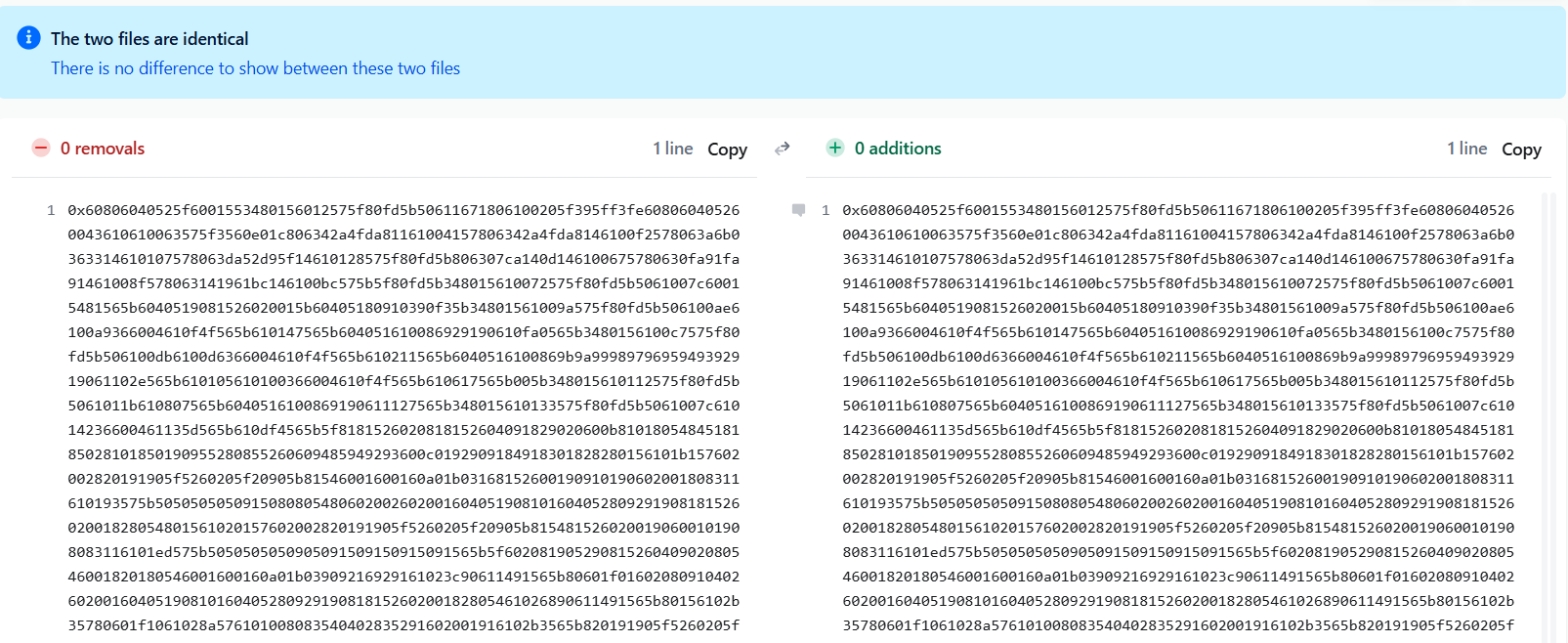


**Gambar 4.x** Bytecode hasil kompilasi smart contract



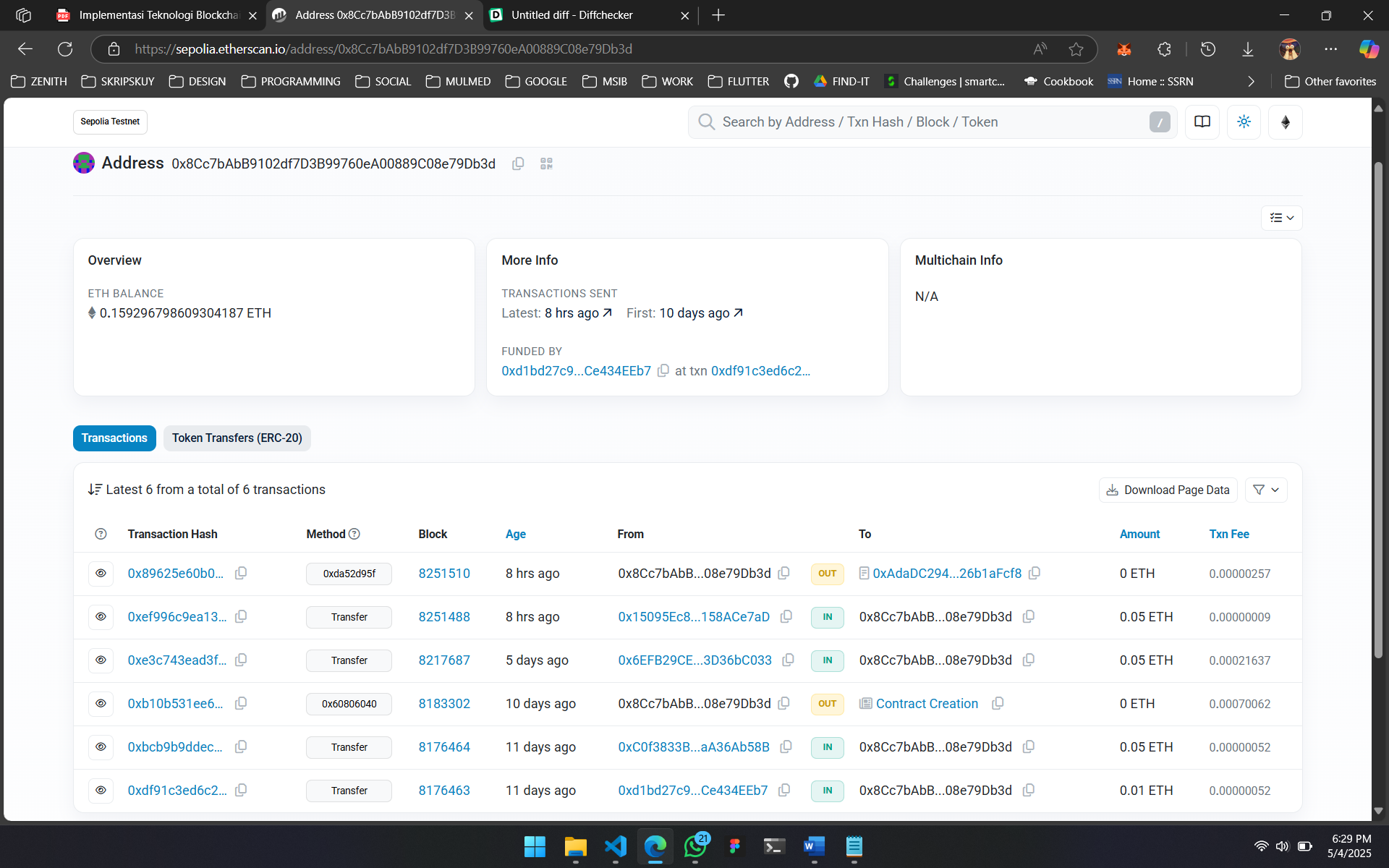
**Gambar 4.x** Bytecode smart contract pada blockchain

Ketika melakukan kompilasi solc dari solidity akan memberikan file json yang berisi data atau kode heksadesimal yang nantinya disimpan pada jaringan blockchain. Pengujian integritas dilakukan untuk memastikan kontrak yang dideploy bersifat identik dengan kontrak yang dikembangkan, sehingga tidak ada modifikasi atau manipulasi selama proses deployment. Bytecode pada computer lokal harus ditambahkan “0x” pada awalannya karena standart hash dari jaringan Ethereum. Berikut adalah hasil komparasi bytecode pada computer lokal dan pada jaringan blockchain.



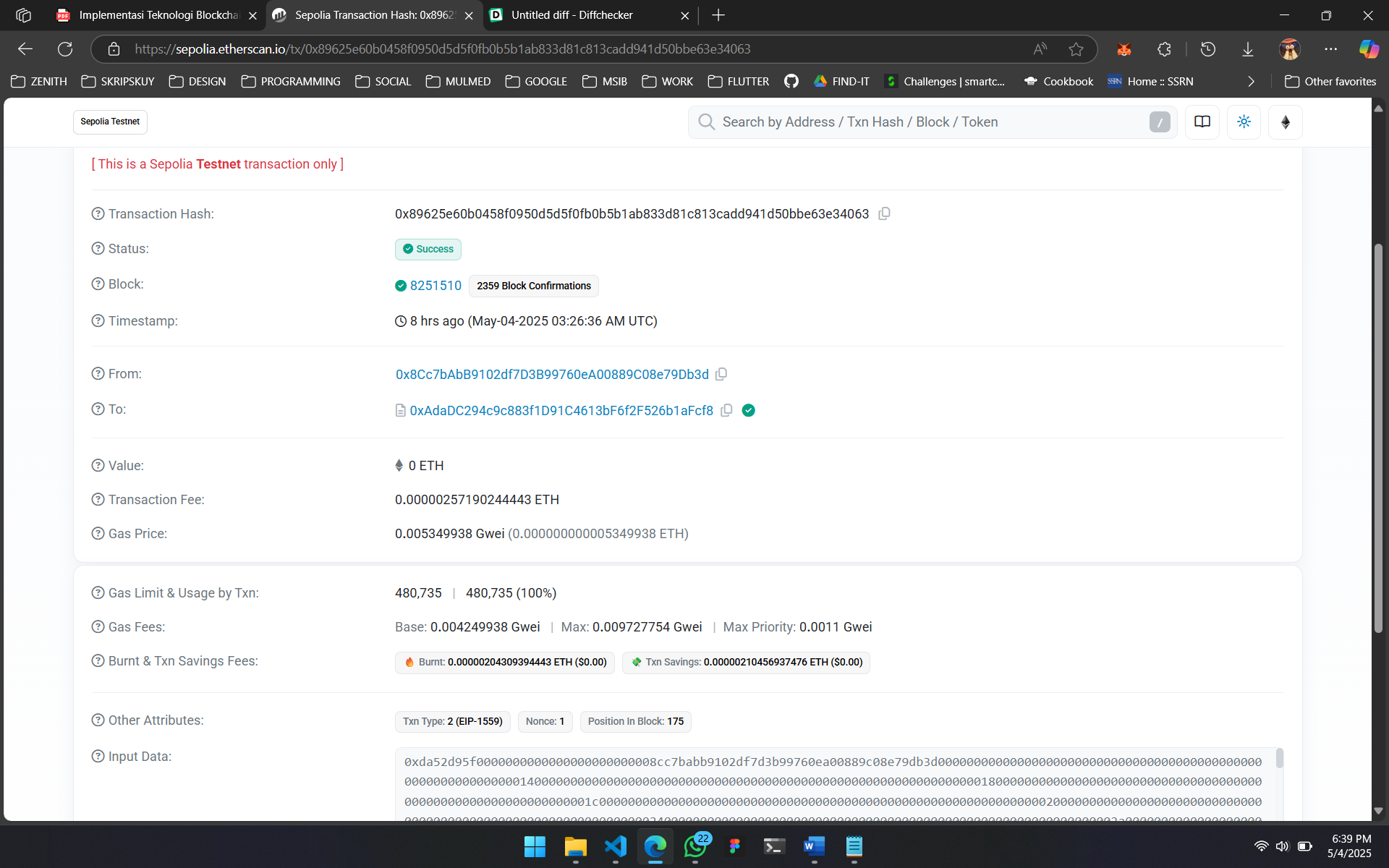
**Gambar 4.x** Hasil perbandingan bytecode

Dari hasil membandingkan bytecode pada lokal dengan yang ada pada jaringan blockhain menggunakan software Diffchecker didapatkan hasil bahwa kedua bytecode bersifat identik, hal ini dapat dipastikan bahwa data yang ada pada lokal itu sama dengan data yang disimpan di blockchain, sehingga integritas dan keamanan data terjaga. Pengujian availability data dilakukan secara onchain dengan mengunjungi sepoliaetherscan dan menginputkan alamat transaksi atau alamat dari wallet yang digunakan transaksi seperti pada gambar dibawah.



**Gambar 4.x** Inteface sepoliaetherscan

Pada sepoliaetherscan ditampilkan data seluruh transaksi yang pernah dilakukan oleh wallet terkait. Data-data tersebut disimpan pada jaringan blockchain Ethereum testnet Sepolia. Selanjutnya dilakukan pengecekan detail transaksi dari transaction hash. Sepoliaetherscan akan menampilkan detail data dari transaksi seperti pada gambar dibawah.



**Gambar 4.x** Inteface sepoliaetherscan detail transaksi

Pada sepoliaetherscan ditampilkan beberapa data seperti, block, address sender, addres receiver, transaction hash, jumlah gasfee dan detail data lainnya. pada sepoliaetherscan juga menampilkan data yang disimpan pada blockchain dalam bentuk yang sudah dienkripsi. Data tersebut akan didekripsi untuk memastikan availability atau transparansi data yang disimpan, kemudian akan dibandingkan dengan data aslinya sebelum disimpan pada blockchain. Hasil dekripsi dan komparasi data adalah sebagai berikut:

**Gambar 4.x** Hasil dekripsi data pada blockchain

**Gambar 4.x** Perbandingan data pada yang disimpan dan sebelum disimpan

**BAB V KESIMPULAN DAN SARAN**

**LAMPIRAN**

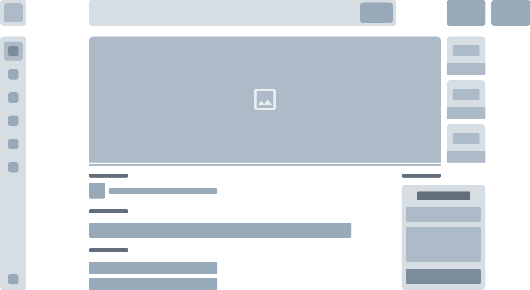
**Lampiran 1.** Wireframe dan Interface halaman dashboard



**Lampiran 2.** Wireframe dan interface halaman donasi



**Lampiran 3.** Wireframe dan interface halaman detail kampanye



**Lampiran 4.** Wireframe dan interface untuk melakukan clain token



**Lampiran 5.** Wireframe dan interface halaman staking

**Lampiran 6**. Kode smart contract crowdfunding

|  |
| --- |
| *// SPDX-License-Identifier: MIT*  pragma solidity ^0.8.9;  contract *Crowdfunding* {      struct *Campaign* {          address owner;          string nameOwner;          string instagram;          string twitter;          string telegram;          string title;          string description;          uint256 target;          uint256 deadline;          uint256 amountCollected;          string image;          address[] donators;          uint256[] donations;      }      mapping(uint256 => Campaign) public campaigns;      uint256 public numberOfCampaigns = 0;      function *createCampaign*(          address *\_owner*,          string memory *\_nameOwner*,          string memory *\_instagram*,          string memory *\_twitter*,          string memory *\_telegram*,          string memory *\_title*,          string memory *\_description*,          uint256 *\_target*,          uint256 *\_deadline*,          string memory *\_image*      ) public returns (uint256) {          require(\_deadline > block.timestamp, "The deadline should be a date in the future");          Campaign storage campaign = campaigns[numberOfCampaigns];          campaign.owner = \_owner;          campaign.nameOwner = \_nameOwner;          campaign.instagram = \_instagram;          campaign.twitter = \_twitter;          campaign.telegram = \_telegram;          campaign.title = \_title;          campaign.description = \_description;          campaign.target = \_target;          campaign.deadline = \_deadline;          campaign.amountCollected = 0;          campaign.image = \_image;          numberOfCampaigns++;          return numberOfCampaigns - 1;      }      function *donateToCampaign*(uint256 *\_id*) public payable {          uint256 amount = msg.value;          Campaign storage campaign = campaigns[\_id];          require(block.timestamp < campaign.deadline, "Campaign has ended");          require(amount > 0, "Donation must be greater than zero");          campaign.donators.*push*(msg.sender);          campaign.donations.*push*(amount);          campaign.amountCollected += amount;          (bool sent,) = payable(campaign.owner).call{value: amount}("");          require(sent, "Failed to send donation to campaign owner");      }        function *getDonators*(uint256 *\_id*) public view returns (address[] memory, uint256[] memory) {          return (campaigns[\_id].donators, campaigns[\_id].donations);      }      function *getCampaigns*() public view returns (Campaign[] memory) {          Campaign[] memory allCampaigns = new Campaign[](numberOfCampaigns);          for (uint256 i = 0; i < numberOfCampaigns; i++) {              Campaign storage item = campaigns[i];              allCampaigns[i] = item;          }          return allCampaigns;      }  } |

**Lampiran 7.** Kode smart contract staking token

|  |
| --- |
| *// SPDX-License-Identifier: MIT*  pragma solidity ^0.8.9;  *// interface erc20*  interface *IERC20* {      function *totalSupply*() external view returns (uint256);      function *balanceOf*(address *account*) external view returns (uint256);      function *transfer*(address *recipient*, uint256 *amount*) external returns (bool);      function *allowance*(address *owner*, address *spender*) external view returns (uint256);      function *approve*(address *spender*, uint256 *amount*) external returns (bool);      function *transferFrom*(address *sender*, address *recipient*, uint256 *amount*) external returns (bool);        event *Transfer*(address indexed *from*, address indexed *to*, uint256 *value*);      event *Approval*(address indexed *owner*, address indexed *spender*, uint256 *value*);  }  *//contract staking token inherit interface erc20*  contract *StakingToken* is *IERC20* {      string public name = "STAKING TOKEN";      string public symbol = "STKN";      uint8 public decimals = 18;      uint256 private \_totalSupply = 1000000000000 ;        mapping(address => uint256) private \_balances;      mapping(address => mapping(address => uint256)) private \_allowances;      constructor() {          \_totalSupply = \_totalSupply \* (10 \*\* uint256(decimals));          \_balances[msg.sender] = \_totalSupply;          emit *Transfer*(address(0), msg.sender, \_totalSupply);      }      function *totalSupply*() external view override returns (uint256) {          return \_totalSupply;      }      function *balanceOf*(address *account*) external view override returns (uint256) {          return \_balances[account];      }      function *allowance*(address *owner*, address *spender*) external view override returns (uint256) {          return \_allowances[owner][spender];      }      function *approve*(address *spender*, uint256 *amount*) external override returns (bool) {          \_allowances[msg.sender][spender] = amount;          emit *Approval*(msg.sender, spender, amount);          return true;      }      function *transfer*(address *recipient*, uint256 *amount*) external override returns (bool) {          require(\_balances[msg.sender] >= amount, "ERC20: transfer amount exceeds balance");          \_balances[msg.sender] -= amount;          \_balances[recipient] += amount;          emit *Transfer*(msg.sender, recipient, amount);          return true;      }      function *transferFrom*(address *sender*, address *recipient*, uint256 *amount*) external override returns (bool) {          require(\_balances[sender] >= amount, "ERC20: transfer amount exceeds balance");          require(\_allowances[sender][msg.sender] >= amount, "ERC20: transfer amount exceeds allowance");            \_balances[sender] -= amount;          \_balances[recipient] += amount;          \_allowances[sender][msg.sender] -= amount;            emit *Transfer*(sender, recipient, amount);          return true;      }  } |

**Lampiran 8.** Kode smart contract reward token

|  |
| --- |
| *// SPDX-License-Identifier: MIT*  pragma solidity ^0.8.9;  *// interface erc20*  interface *IERC20* {      function *totalSupply*() external view returns (uint256);      function *balanceOf*(address *account*) external view returns (uint256);      function *transfer*(address *recipient*, uint256 *amount*) external returns (bool);      function *allowance*(address *owner*, address *spender*) external view returns (uint256);      function *approve*(address *spender*, uint256 *amount*) external returns (bool);      function *transferFrom*(address *sender*, address *recipient*, uint256 *amount*) external returns (bool);        event *Transfer*(address indexed *from*, address indexed *to*, uint256 *value*);      event *Approval*(address indexed *owner*, address indexed *spender*, uint256 *value*);  }  *//contract staking token inherit interface erc20*  contract *RewardToken* is *IERC20* {      string public name = "REWARD TOKEN";      string public symbol = "RTKN";      uint8 public decimals = 18;      uint256 private \_totalSupply = 1000000000000 ;        mapping(address => uint256) private \_balances;      mapping(address => mapping(address => uint256)) private \_allowances;      constructor() {          \_totalSupply = \_totalSupply \* (10 \*\* uint256(decimals));          \_balances[msg.sender] = \_totalSupply;          emit *Transfer*(address(0), msg.sender, \_totalSupply);      }      function *totalSupply*() external view override returns (uint256) {          return \_totalSupply;      }      function *balanceOf*(address *account*) external view override returns (uint256) {          return \_balances[account];      }      function *allowance*(address *owner*, address *spender*) external view override returns (uint256) {          return \_allowances[owner][spender];      }      function *approve*(address *spender*, uint256 *amount*) external override returns (bool) {          \_allowances[msg.sender][spender] = amount;          emit *Approval*(msg.sender, spender, amount);          return true;      }      function *transfer*(address *recipient*, uint256 *amount*) external override returns (bool) {          require(\_balances[msg.sender] >= amount, "ERC20: transfer amount exceeds balance");          \_balances[msg.sender] -= amount;          \_balances[recipient] += amount;          emit *Transfer*(msg.sender, recipient, amount);          return true;      }      function *transferFrom*(address *sender*, address *recipient*, uint256 *amount*) external override returns (bool) {          require(\_balances[sender] >= amount, "ERC20: transfer amount exceeds balance");          require(\_allowances[sender][msg.sender] >= amount, "ERC20: transfer amount exceeds allowance");            \_balances[sender] -= amount;          \_balances[recipient] += amount;          \_allowances[sender][msg.sender] -= amount;            emit *Transfer*(sender, recipient, amount);          return true;      }  } |

**Lampiran 9.** Kode smart contract token drop

|  |
| --- |
| *// SPDX-License-Identifier: MIT*  pragma solidity ^0.8.9;  import "@openzeppelin/contracts/extension/Ownable.sol";  interface *IERC20* {      function *totalSupply*() external view returns (uint256);      function *balanceOf*(address *account*) external view returns (uint256);      function *transfer*(address *recipient*, uint256 *amount*) external returns (bool);      function *allowance*(address *owner*, address *spender*) external view returns (uint256);      function *approve*(address *spender*, uint256 *amount*) external returns (bool);      function *transferFrom*(address *sender*, address *recipient*, uint256 *amount*) external returns (bool);      event *Transfer*(address indexed *from*, address indexed *to*, uint256 *value*);      event *Approval*(address indexed *owner*, address indexed *spender*, uint256 *value*);  }  contract *TokenDrop* is *Ownable* {      IERC20 public stakingToken;      mapping(address => bool) public hasClaimed; *// Menyimpan status klaim token*      event *Claimed*(address indexed *user*, uint256 *amount*);      constructor(address *\_stakingToken*) {          stakingToken = *IERC20*(\_stakingToken);      }  *// Fungsi untuk klaim staking token*      function *claimToken*() external {          require(!hasClaimed[msg.sender], "You have already claimed");          hasClaimed[msg.sender] = true;          stakingToken.*transfer*(msg.sender, 1 \* (10 \*\* 18));          emit *Claimed*(msg.sender, 1 \* (10 \*\* 18));      }      function *depositTokens*(uint256 *\_amount*) external *onlyOwner* {          stakingToken.*transferFrom*(msg.sender, address(this), \_amount);      }      function *withdrawTokens*(uint256 *\_amount*) external *onlyOwner* {          stakingToken.*transfer*(msg.sender, \_amount);      }      function *getDepositedSupply*() external view returns (uint256) {          return stakingToken.*balanceOf*(address(this));      }  } |

**Lampiran 10.** Kode smart contract staking

|  |
| --- |
| *// SPDX-License-Identifier: MIT*  pragma solidity ^0.8.9;  contract *Staking* {      address public rewardToken;      address public stakingToken;      address public admin;      uint256 public rewardTokenBalance;      uint256 public stakingTokenBalance;      uint80 public timeUnit;      uint256 public rewardRatioNumerator;      uint256 public rewardRatioDenominator;      mapping(address => uint256) public stakedBalances;      mapping(address => uint256) public stakingTimestamps;        event *Staked*(address indexed *user*, uint256 *amount*);      event *Unstaked*(address indexed *user*, uint256 *amount*);      event *RewardClaimed*(address indexed *user*, uint256 *amount*);      event *RewardDeposited*(uint256 *amount*);      event *RewardWithdrawn*(uint256 *amount*);      modifier *onlyAdmin*() {          require(msg.sender == admin, "Not authorized");          \_;      }      constructor(          address *\_rewardToken*,          address *\_stakingToken*,          uint80 *\_timeUnit*,          uint256 *\_rewardRatioNumerator*,          uint256 *\_rewardRatioDenominator*      ) {          admin = msg.sender;          rewardToken = \_rewardToken;          stakingToken = \_stakingToken;          timeUnit = \_timeUnit;          rewardRatioNumerator = \_rewardRatioNumerator;          rewardRatioDenominator = \_rewardRatioDenominator;      }      function *stake*(uint256 *\_amount*) external {          require(\_amount > 0, "Cannot stake 0");          stakedBalances[msg.sender] += \_amount;          stakingTimestamps[msg.sender] = block.timestamp;          stakingTokenBalance += \_amount;          emit *Staked*(msg.sender, \_amount);      }      function *unstake*(uint256 *\_amount*) external {          require(stakedBalances[msg.sender] >= \_amount, "Insufficient staked balance");          stakedBalances[msg.sender] -= \_amount;          stakingTokenBalance -= \_amount;          emit *Unstaked*(msg.sender, \_amount);      }      function *claimRewards*() external {          require(stakedBalances[msg.sender] > 0, "No staked tokens");          uint256 reward = *calculateReward*(msg.sender);          require(reward <= rewardTokenBalance, "Not enough rewards");          rewardTokenBalance -= reward;          stakingTimestamps[msg.sender] = block.timestamp;          emit *RewardClaimed*(msg.sender, reward);      }      function *depositRewardTokens*(uint256 *\_amount*) external *onlyAdmin* {          require(\_amount > 0, "Amount must be greater than zero");          rewardTokenBalance += \_amount;          emit *RewardDeposited*(\_amount);      }      function *withdrawRewardTokens*(uint256 *\_amount*) external *onlyAdmin* {          require(\_amount <= rewardTokenBalance, "Not enough reward tokens");          rewardTokenBalance -= \_amount;          emit *RewardWithdrawn*(\_amount);      }      function *calculateReward*(address *\_user*) internal view returns (uint256) {          uint256 stakingDuration = block.timestamp - stakingTimestamps[\_user];          return (stakedBalances[\_user] \* stakingDuration \* rewardRatioNumerator) /              (timeUnit \* rewardRatioDenominator);      }      function *getStakeInfo*(address *\_user*) external view returns (uint256 *stakedAmount*, uint256 *rewardAmount*) {          stakedAmount = stakedBalances[\_user];          rewardAmount = *calculateReward*(\_user);      }  } |