

**IMPLEMENTASI SMART CONTRACT PADA PLATFORM CROWDFUNDING BERBASIS BLOCKCHAIN ETHEREUM MENGGUNAKAN ALGORITMA KONSENSUS PROOF OF STAKE**

**SKRIPSI**

Diajukan sebagai salah satu syarat penelitian untuk memenuhi kelulusan program studi S1 Teknologi Informasi Fakultas Ilmu Komputer Universitas Jember

Oleh:

**Muhammad Amanda Maulana Malik Ibrahim**

**212410102035**

**KEMENTRIAN PENDIDIKAN TINGGI, SAINS DAN TEKNOLOGI**

**UNIVERSITAS JEMBER**

**FAKULTAS ILMU KOMPUTER**

**PROGRAM STUDI S1 TEKOLOGI INFORMASI**

**2025**

**PERSEMBAHAN**

Puji Syukur kehadirat Allah SWT yang telah melimpahkan Rahman serta hidayah-Nya sehingga penulis dapat menyelesaikan skripsi ini dengan penuh kesabaran dan kerendahan hati yang luar biasa. Keberhasilan dalam penyelesaian skripsi ini tentunya tidak lepas dari bantuan berbagai pihak. Oleh karena itu penulis menyampaikan terima kasih kepada:

1. Allah SWT yan gsenantiasa memerikan rahmat serta hidayah-Nya dan memberikan kemudahan kepada penulis dalam menyelesaikan skripsi
2. Ayahanda Muhammad Bahrur Razi dan Ibunda Siti Khumairoh yang selalu memerikan dukungan penulis berupa moral maupun materil yang tak terhingga, serta doa yang tiada putusnya yang selalu diberikan kepada penulis sehingga penulis mampu menyelesaikan studi sarjana hingga selesai.
3. Bapak Diksy Media Firmansyah S.Kom., M.Kom dan Bapak Yanuar Nurdiansyah S.T., M.Cs selaku dosen pembimbing yang telah mendukung, memeberi nasehat dan membantu penulis untuk menyelesaikan dan menyempurnakan skripsi ini.
4. Keluarga besar Bapak Alm. H. Moh. Amin dan Ibu Hj. Siti Susmiyah yang selalu memberikan doa serta dukungan dalam proses Menyusun skripsi.
5. Kepada pemilik NIM 1034, Terima kasih telah menjadi bagian dari perjalanan yang penuh rintangan ini. Berkontribusi banyak dalam proses penyusunan skripsi ini baik waktu, tenaga, doa dan dukungan yang telah di berikan.
6. Terimakasih kepada teman-teman Fakultas Ilmu Komputer Universitas Jember dan penghuni indekost putra Dwi Nandana 1 yang telah menemani penulis selama masa perkuliahan, masa di *land of dawn*, dan masa ketika meratakan balai kota, hingga sampai skripsi ini selesai.

**MOTTO**

*“Sesungguhnya beserta kesulitan ada kemudahan, apabila engkau telah selsai (dengan suatu kebaikan), teruslah bekerja keras (untuk kebaikan yang lain), dan hanya kepada Tuhanmu kamu berharap”*

(Q.S Al-Insyirah: 5 -8)

“Survive The Dip, Enjoy the Pump, Always DYOR, In Decentralization We Trust”

(Me)

“PALEEEEEEEEEEEEEEEEE!”

(Warden beban)

**PERNYATAAN ORISINALITAS**

Saya yang bertanda tangan di bawah ini :

Nama: Muhammad Amanda Maulana Malik Ibrahim

NIM: 212410102035

Menyatakan dengan sesungguhnya karya ilmiah yan gberjudul “Implementasi *Smart Contract* Pada Platform *Crowdfunding* Berbasis Blockchain Ethereum Menggunakan Algoritma Konsensus Proof of Stake” adalah benar-benar hasil karya sendiri, kecuali jika dalam pengutipaan substansi yang disebutkan sumbernya, belum pernah diajukan pada institusi manapun, dan bukan karya jiplakan. Saya bertanggung jawab atas keabsahan dan kebenaran isinya sesuai dengan sikap ilmiah yang harus dijunjung tinggi.

Demikian pernyataan ini saya buat dengan sebenarnya, tanpa ada tekanan dan paksaan dari pihak manapun serta bersedia mendapat sanksi akademik jika di kemudian hari pernyataan ini tidak benar.

Jember, XX XXXX 2025

Yang menyatakan,

M Amanda Maulana Malik I

212410102035

**HALAMAN PERSETUJUAN**

Skripsi berjudul Implementasi *Smart Contract* Pada Platform *Crowdfunding* Berbasis Blockchain Ethereum Menggunakan Algoritma Konsensus Proof of Stake” telah diuji oleh Fakultas Ilmu Komputer Universitas Jember pada:

Hari :

Tanggal :

Tempat : Fakultas Ilmu Komputer Universitas Jember

Pembimbing Tanda Tangan

1. Pembimbing Utama

Nama : Diksy Media Firmansyah, S.Kom., M.Kom. (……………………)

NIP : 199112132023211015

2. Pembimbing Anggota

Nama : Yanuar Nurdiansyah, S.T., M.Cs. (……………………)

NIP : 198201012010121004

Penguji

1. Penguji Utama

Nama : Prof. Drs. Antonius Cahya Prihandoko, M.App.Sc. Ph.D (……………………)

NIP :

2. Penguji Anggota

Nama : Mohammad Zarkasi, S.Kom., M.Kom (……………………)

NIP :

**ABSTRAK**

**ABSTRACT**

**RINGKASAN**

**PRAKATA**

Puji Syukur kehadirat Allah SWT yang telah melimpahkan rahmat serta hidayah-Nya sehingga penulis dapat menyelesaikan skripsi yang berjudul **“Implementasi Smart Contract Pada Platform Crowdfunding Berbasis Blockchain Ethereum Menggunakan Algoritma Konsensus Proof of Stake”** dengan penuh kerendahan hati dan kesabaran yang luar biasa.

Keberhasilan dalam penulisan skripsi ini tentunya tidak lepas dari berbagai pihak. Oleh karena itu penulis menyampaikan terima kasih kepada:

1. Prof. Drs. Antonius Cahya Prihandoko, M.App.Sc. Ph.D. Selaku Dekan Fakultas Ilmu Komputer Universitas Jember
2. (Kaprodi TI). Selaku ketua program studi S1 Teknologi Informasi
3. (DPA). Selaku dosen wali penulis
4. (DPU, DPA). Yang telah membimbing, mendukung, dan memberi nasihat sehingga bisa menyelesaikan skripsi.
5. (penguji). Selaku penguji satu dan penguji 2 skripsi penulis
6. Para dosen, staff, dan seluruh civitas akademika FASILKOM UNEJ yang telah meberikan semangat, dukungan dan fasilitas selama perkuliahan.

Jember, XX XXXX 2025

M Amanda Maulana Malik I

212410102035

**DAFTAR ISI**

**DAFTAR TABEL**

**DAFTAR GAMBAR**

**DAFTAR LAMPIRAN**

**BAB I**

**PENDAHULUAN**

**1.1 Latar Belakang**

Crowdfunding berasal dari kata "crowd" yang berarti keramaian atau sekelompok orang, dan "funding" yang berarti pengumpulan dana. Crowdfunding adalah metode pengumpulan dana dari banyak orang yang memiliki minat terhadap suatu kegiatan atau bisnis. Crowdfunding memberikan peluang bagi kreator untuk mendapatkan dana dari masyarakat yang tertarik dengan gagasan mereka dan menciptakan potensi untuk mempercepat pengembangan bisnis atau kegiatan yang sedang berjalan. (Aufila et al., 2024)

Sistem crowdfunding tradisional sudah memberikan fungsional dengan baik, namun masih terdapat beberapa tantangan dalam sistem terpusat, terutama terkait dengan kepercayaan dan keamanan. Funder sering kali kesulitan menentukan apakah kampanye pada platform crowdfunding benar-benar sah, funder juga bisa saja dihadapkan pada kasus penipuan, kampanye yang tidak memenuhi syarat sebagai penerima dana, dan risiko penyalahgunaan dana. Selain itu, peran perantara pihak ketiga dalam proses ini menyebabkan biaya operasional yang tinggi. (Chatkar et al., 2023)

Salah satu solusi untuk mengatasi hal ini adalah dengan menerapkan teknologi blockchain. Blockchain, sebagai teknologi desentralisasi, menawarkan tingkat keamanan dan transparansi yang tinggi. Teknologi ini berfungsi sebagai sistem pencatatan yang terdistribusi dan terdesentralisasi, di mana data disimpan dalam blok-blok yang saling terhubung dan terenkripsi serta terverifikasi oleh banyak node. Proses verifikasi ini memastikan bahwa data hampir mustahil dimanipulasi tanpa persetujuan mayoritas jaringan. Dengan demikian, blockchain menjadi solusi potensial untuk meningkatkan kepercayaan, keamanan dan transparansi dalam sistem penggalangan dana. (Hasan et al., 2024)

Dalam era perkembangan teknologi blockchain terdapat teknologi smart contract yang bersifat objektif untuk menentukan secara spesifik bagaimana proses transaksi akan dikelola dan tindakan apa yang akan diambil ketika suatu peristiwa terjadi. Seluruh data dan transaksi dapat divalidasi secara otomatis, dan dieksekusi oleh kode dalam jaringan blockchain sebagai perjanjian digital, Kontrak ini disimpan dalam blockchain dan didistribusikan ke semua node, sehingga tidak bisa diubah, selayaknya data pada blockchain yang bersifat permanen dan transparan. (Hermawan et al., 2023)

Teknologi blockchain terus berkembang dari tahun ke tahun melalui berbagai cara, salah satunya melalui implementasi smart contract dengan berbagai konsensus tergantung pada jaringan blockchainnya salah satunya adalah Proof of Stake (PoS). Teknologi blockchain sebagai fundamental di balik mata uang kripto ini telah merevolusi cara pengelolaan dan pemahaman terhadap aset digital. (Julio et al., 2024)

Penelitian terdahulu yang dilakukan oleh (Baihaqsani, 2023) dengan mengimplementasikan smart contract blockchain pada sistem klaim asuransi. Peelitian ini mendapat hasil bahwa blockchain dapat meningkatkan transparansi, keamanan dan efisiensi pada proses klaim, serta mencegah manipulasi data melalui smart contract menggunakan algoritma Proof of Work (PoW). Penelitian lainnya yang dilakukan oleh (Sahputra, 2019) dengan mengimplementasikan smart contract pada sistem voting. Penelitian ini berhasil memastikan integritas data yang sudah di hash dan disimpan pada blockchain melalui pengecekan pada panel administrator.

Beberapa penelitian terdahulu telah membuktikan bahwa permasalahan pada data yang bersifat tetap dan transaksional seperti data asuransi dan voting terdapat pada transparansi, dan integritas data yang semua permasalahan tersebut bisa diselesaikan menggunakan teknologi blockchain. Berdasarkan latar belakang tersebut, penulis mengusulkan untuk mengimplementasikan smart contract blockchain pada sistem crowdfunding, diharapkan sistem yang dikembangkan dapat menjamin keamanan dan transparansi data. Diharapkan penggalang dana dapat lebih mudah membangun kepercayaan dengan donator melalui transparansi penuh dalam transaksi, sementara penerima dana akan mendapatkan proses pencairan yang lebih cepat dan aman tanpa keterlibatan pihak ketiga yang kompleks.

**1.2 Rumusan Masalah**

Berdasarkan uraian latar belakang diatas maka didapatkan rumusan masalah dalam penelitian adalah sebagai berikut:

1. Bagaimana proses perancangan dan implementasi smart contract untuk platform crowdfunding berbasis blockchain Ethereum yang aman dan efisien dengan menggunakan konsensus Proof of Stake (PoS)?
2. Bagaimana mekanisme dan hasil pengujian keamanan transaksi dalam smart contract tersebut ketika digunakan pada platform crowdfunding berbasis blockchain Ethereum?

**1.3 Tujuan Penelitian**

Dengan masalah yang sudah disebutkan, tujuan dari penelitian ini yaitu:

1. Menghasilkan smart contract yang dapat menangani proses crowdfunding dan menjamin keamanan data transaksi pada aplikasi berbasis blockchain Ethereum menggunakan konsensus Proof of Stake (PoS)
2. Menganalisis hasil pengujian smart contract untuk memastikan keamanan dan integritas data transaksi pada aplikasi berbasis blockchain Ethereum menggunakan konsensus Proof of Stake (PoS)

**1.4 Manfaat Penelitian**

Penelitian ini diharapkan dapat memberikan manfaat bagi beberapa pihak antara lain:

1. Bagi Akademisi dan peneliti

Menambah wawasan dan pengetahuan mengenai penerapan teknologi blockchain dalam sisi keamanan data pada platform crowdfunding menggunakan algoritma Proof of Stake, serta membantu memberikan referensi penelitian lebih lanjut mengenai penerapan teknologi blockchain.

1. Bagi Funder dan Fundraiser

Penelitian ini diharapkan memberikan wawasan yang bermanfaat bagi pemberi dana (funder) dan penerima dana (fundraiser) mengenai penerapan teknologi blockchain pada crowdfunding, sehingga funder dapat memastikan dana yang mereka berikan diterima dan dikelola dengan baik dan tidak disalahgunakan. Sedangkan fundraiser dapat membangun kepercayaan yang lebih melalui sistem yang lebih transparan dan terdesentralisasi

**1.5 Batasan Masalah**

Dalam penelitian ini, penulis memberikan beberapa Batasan masalah untuk menghindari terjadi penyimpangan selama proses penelitian dan penulisan. Beberapa Batasan masalah antara lain:

1. Penelitian tidak membahas aspek off-chain dari platform crowdfunding, meliputi manajemen pengguna dan UI/UX aplikasi.
2. Analisis smart contract mencakup aspek keamanan, fungsionalitas dan non- fungsioanlitas tanpa membandingkan dengan mekanisme crowdfunding konvensional.
3. Pengujian smart contract crowdfunding dilakukan pada jaringan testnet Sepolia Ethereum, tanpa implementasi di mainnet Ethereum.

**BAB II**

**TINJAUAN PUSTAKA**

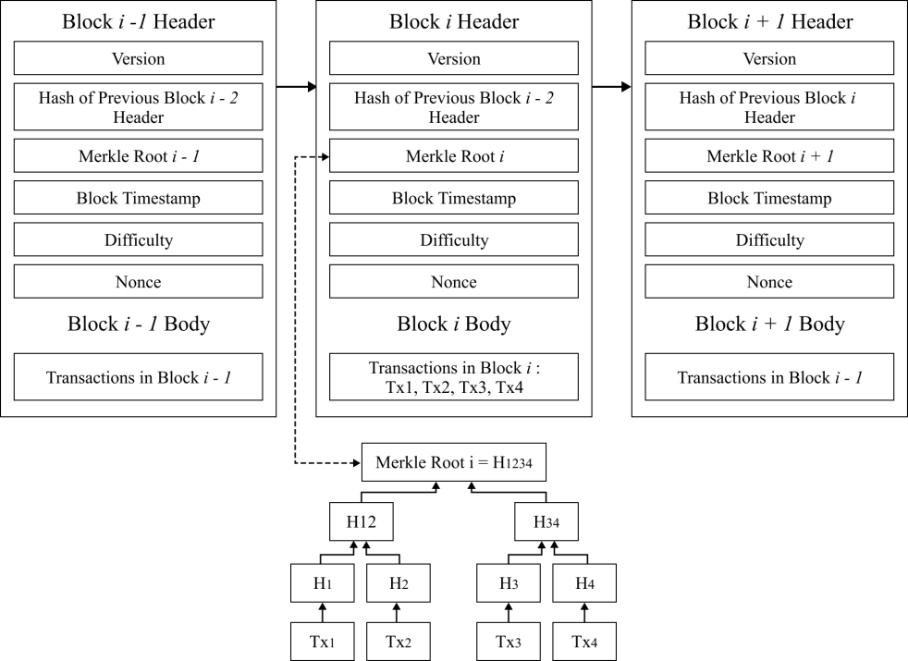
**2.1 Penelitian Terdahulu**

Beberapa penelitian terdahulu yang berkaitan dengan penelitian penulis ini akan disajikan pada table 1 sebagai berikut:

|  |  |
| --- | --- |
|  |  |

**2.2 Blockchain**

Blockchain adalah teknologi terdesentralisasi yang memungkinkan penyimpanan data secara aman, transparan, dan tidak dapat diubah melalui jaringan yang tersebar atau terdesentralisasi. Dalam blockchain, data disimpan dalam buku besar terdistribusi. Teknologi blockchain menyediakan integritas dan ketersediaan yang memungkinkan pengguna dalam jaringan blockchain untuk menulis, membaca, dan memverifikasi transaksi yang tercatat dalam buku besar terdistribusi. Namun, teknologi ini tidak memungkinkan penghapusan atau modifikasi terhadap transaksi dan informasi lain yang tersimpan di buku besar tersebut. Sistem blockchain didukung dan diamankan oleh elemen-elemen kriptografi dan enkripsi, seperti tanda tangan digital, fungsi hash, dan sebagainya. Elemen-elemen ini menjamin bahwa transaksi yang dicatat dalam buku besar dilindung integritasnya, diverifikasi keasliannya, dan tidak dapat diubah (Huaqun Guo, 2022)



**Gambar 1**. Struktur Blockchain

Pada struktur blockchain di pisah menjadi dua bagian yaitu block header dan block body, block header berisi mengenai metadata block yang terdiri dari beberapa bagian yakni:

Bagian Header Block:

1. Version: Versi blockchain
2. Hash of Previous Block: Hash dari header block sebelumnya
3. Merkle Root: Hash dari semua transaksi dalam block
4. Timestamp: Waktu pembuatan block
5. Difficulty: Tingkat kesulitan untuk menambang di block ini Nonce: Nilai yang diubah selama proses penambangan untuk menemukan hash yang sesuai

Sedangkan pada bagian block body berisi semua data transaksi yang sudah terenkripsi di dalam block tersebut.

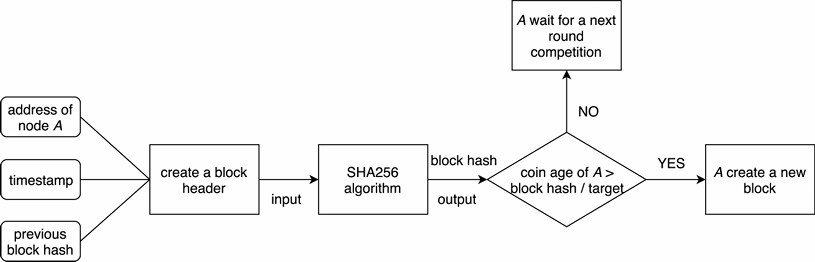
**2.3 Smart Contract**

Smart contract adalah program yang berjalan di atas blockchain yang secara otomatis mengeksekusi, mengelola, atau mendokumentasikan peristiwa sesuai dengan syarat dan ketentuan yang telah diprogramkan. Smart contract memungkinkan pelaksanaan kode yang objektif dalam mengatur proses secara otomatis. Salah satu tujuan utama pengembangan smart contract di Ethereum adalah mengatasi keterbatasan Bitcoin. Smart contract menawarkan otonomi, efisiensi, akurasi, serta penghematan biaya, tanpa memerlukan perantara. Selain itu, smart contract tidak harus melibatkan dua pihak atau bersifat mengikat secara hukum (Huaqun Guo, 2022)

**2.4 Konsensus Proof of Stake**

Algorima konsensus adalah algoritma yang berfungsi sebagai mekanisme utama pada jaringan blockhain yang bertugas untuk mencapai kesepakatan pada smart contract di dalam jaringan blockchain. Konsensus berjalan dengan memanfaatkan sumberdaya mayoritas pengguna untuk menjaga integritas sistem. Konsensus memastikan bahwa transaksi yang dicatat di blockchain tidak berbahaya, tidak duplikat dan tidak double spending. Mekanisme konsensus menjadi inti dari seluruh kegiatan di blockchain,

Algoritma konsensus proof of Stake (PoS) dikembangkan untuk mengatasi konsumsi daya yang besar pada Proof of Work (PoW). Dalam Proof of Stake pengguna tidak perlu memecahkan masalah matematika untuk mencapai consensus (nonce), melainkan cukup menggunakan beberapa cryptocurrency untuk di taruhkan (Staking). Pencipta blok baru dipilih secara acak dari uang pengguna yang telah melakukan staking, tanpa seorang pun dapat memprediksi gilirannya. Sistem ini mengurangi pemborosan sumber daya PoW dan mendorong pemegang koin untuk meningkatkan waktu penyimpanan, yang meningkatkan keamanan blockchain. (Jannah Yusoff, 2022)



**Gambar 2.** Alur Algoritma Proof of Stake

**2.5 Kriptografi**

Kriptografi adalah seni dan ilmu yang digunakan untuk menyembunyikan pesan, dengan tujuan menjaga kerahasiaan informasi. Pesan yang belum dienkripsi disebut plaintext, sedangkan setelah proses enkripsi, pesan tersebut menjadi ciphertext. Kriptografi mempelajari dan menerapkan teknik komunikasi yang aman dari ancaman pihak ketiga.

Komunikasi yang aman memastikan bahwa pesan atau data yang ditransfer antara dua pihak tidak dapat diakses oleh pihak lawan. Dalam dunia kriptografi, lawan adalah entitas yang berusaha mendapatkan informasi berharga dan merusak prinsip-prinsip keamanan. Kerahasiaan data, integritas data, autentikasi, dan non-repudiasi adalah prinsip dasar dalam kriptografi modern. Kekuatan kriptografi diukur dari waktu dan sumber daya yang diperlukan untuk mengembalikan plaintext. Hasil dari kriptografi yang kuat adalah ciphertext yang sangat sulit dipecahkan tanpa alat dekripsi yang tepat, bahkan dengan semua kekuatan komputasi yang ada saat ini (Archana B U, 2023)

**2.6 Ethereum**

Ethereum adalah cryptocurrency terbesar kedua berdasarkan kapitalisasi pasar. Platform ini merupakan sistem komputasi terdistribusi open source yang dirancang untuk mendukung smart contract, memungkinkan pengembang membangun aplikasi terdesentralisasi dengan mudah dengan tekbologi blockchain. Berbeda dengan Bitcoin yang hanya mencatat transaksi antar alamat, blockchain Ethereum juga menyimpan alamat dengan kode yang dapat dijalankan oleh Ethereum Virtual Machine (EVM).

Transaksi pada blockchain Ethereum mencakup data input untuk program yang kemudian diproses oleh EVM menggunakan bahasa pemrograman yang sesuai. Dasar utama dari sistem Ethereum adalah akun, yang diperlukan untuk mengirimkan transaksi. Terdapat dua jenis akun di Ethereum: Akun yang Dimiliki Secara Eksternal (Externally Owned Accounts/EOA), yang digunakan langsung oleh pengguna untuk transaksi, dan Akun Kontrak (Contract Accounts), yang memungkinkan interaksi dengan kontrak lain serta transaksi internal. Setiap akun memiliki pasangan kunci, yaitu kunci pribadi dan kunci publik. Pemahaman mengenai perbedaan antara kunci pribadi, transaksi, pengirim, serta penggunaan nilai hash sangat penting untuk melacak transaksi di blockchain. (Dzulfikar & Susanto, 2020)

**2.7 Decentralized Apps (Dapps)**

Decentralized App atau Dapps adalah program berbasis blockchain yang berjalan pada infrastruktur jaringan peer-to-peer, terdesentralisasi dan tidak terpusat pada suatu pihak manapun. Jaringan peer-to-peer adalah jaringan di mana dua pengguna berinteraksi atau berbagi informasi tanpa intervensi pihak pusat. Dalam Dapps semua data transaksi dilindungi dengan kriptografi yang berfungsi untuk mengamankan komunikasi anatara dua block dari sumber pihak ketiga. Dapps umumnya menggunakan uang kripto internal sebagai pendorong utama ekosistemnya dan telah diterapkan dalam berbagai bidang, seperti energi, asuransi, dan keuangan. Perbandingan antara aplikasi blockchain terpusat dan terdesentralisasi menunjukkan perbedaan signifikan dalam keamanan data. Berikut ini adalah beberapa perbedaan dari centralized app (Apps) dan Dapps. (Raza et al., 2024)

|  |  |
| --- | --- |
| **Centralized App** | **Decentralized App** |
| Apps beroperasi menggunakan sebuah arsitektur *client-server* | Dapps menggunakan smart contract untuk berinteraksi dengan *client* |
| Database pengguna dapat diakses dan terhubung pada backend server | Menggunakan jaringan peer-to-peer pada backend |
| Membutuhkan arahan dari superadmin (pusat) untuk menetapkan *role* dan *permission* | Tidak ada arahan dari pihak superadmin (pusat) |
| Kerentanan terhadap keamanan dan privasi | Aman, Tidak tapat diubah dan dapat diatur secara mandiri |

**Tabel 2.** Perbandingan Centralized App dan Decentralized App

**2.8 Thirdweb**

Thirdweb merupakan sebuah tools dan utility yang cukup lengkap untuk membatu para developer Web3/Dapps untuk berinteraksi dengan berbagai jaringan blockchain, seperti Ethereum yang langsung dari proyek Javascript. Tools ini menyediakan integrasi fitur blockchain ke aplikasi front-end menjadi lebih mudah (Kolomoyets & Kynash, 2023).

**2.9 Hardhat JS**

Hardhat adalah framework berbasis Node.js yang dikembangkan oleh Nomic Fundation yang memilii banyak fitur, seperti dukungan pengembangan, pengujian menggunakan Javascript dan juga proses deploy smart contract. Hardhat memiliki kelebihan yang sangat fleksibel dan bisa dikonfigurasi sesuai dengan kebutuhan. Para pengembang bisa mengatur berbagai bagian dari siklus pengembangan proyek, sehingga memiliki kontrol yang detail terhadap bagaimana framework ini bekerja (Helms & McGahon, 2023).

**2.10 Mythril**

Mythril adalah salah satu framework paling matang untuk analisis smart contract di blockchain yang dirancang khusus untukEthereum Virtual Machine (EVM). Mythril menggunakan teknik symbolic execution, yaitu teknik yang mengevaluasi perilaku program dengan menggunakan input simbolik (bukan nilai tetap) untuk menelusuri semua kemungkinan eksekusi kerentanan dalam kontrak pintar. Mythril lebih unggul dibandingkan dengan analisis statis ringan (lightweight static analysis), Mythril cenderung menghasilkan lebih sedikit alert, tetapi mampu menunjukkan kemungkinan nyata dari serangan yang teridentifikasi (Bonomi et al., 2023).

**BAB III**

**METODOLOGI PENELITIAN**

**3.1 Jenis Penelitian**

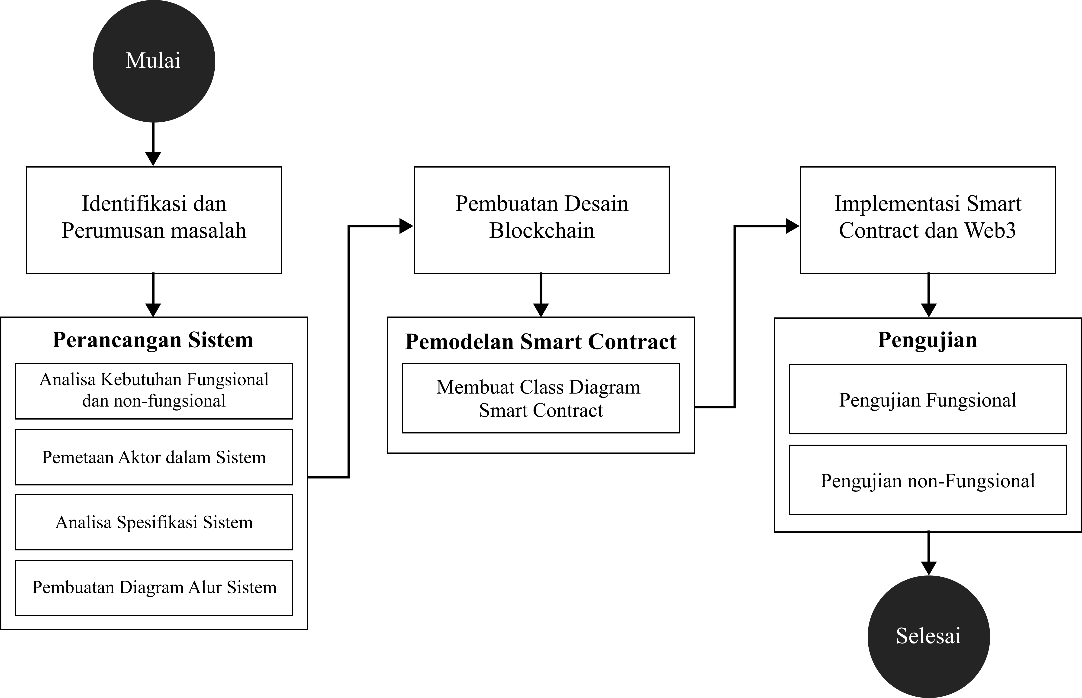
Metode dalam penelitian ini merupakan penelitian pengembangan (RnD). Penelitian pengembangan diawali dengan proses analisis terhadap permasalahan yang diidentifikasi dan diatasi dengan mengembangkan sebuah model atau produk. Produk atau model yang dikembangkan dirancang untuk menjawab permasalahan yang ada. (Maruwu, 2024).

**3.2 Objek Penelitian**

Objek dari penelitian ini adalah smart contract yang nantinya akan di deploy pada jaringan blockchain dan konsensus Proof of Stake. Smart contract yang dihasilkan kemudian akan digunakan untuk menangani proses transaksi pada platform crowdfunding.

**3.3 Tahapan Penelitian**

Pada bagian ini berisi tahapan yang dilakukan peneliti selama penelitian berlangsung, berikut beberpa tahapan penelitian ditunukkan pada gambar G.5.



**Gambar 5**. Tahapan Penelitian

**3.3.1 Identifikasi dan Perumusan Masalah**

Identifikasi masalah dilakukan melalui pencarian berita, kejadian di lingkungan sekitar dan di internet untuk mengetahui permasalahan yang terjadi. Berdasarkan beberapa kasus yang ditemukan, seperti dugaan penyelewengan dana oleh Yayasan Rumah Penghafal Qur’an (RFPS) pada tahun 2022, pengelola dana terduga hanya menyalurkan sebagian kecil dana donasi berkisar antara Rp 300.000 (tiga ratus ribu rupiah) hingga Rp 50.000.000 (lima puluh juta rupiah) dari total donasi RP 1,6 miliar (jawapos.com, 2022). Terdapat juga kasus lembaga filantropi Aksi Cepat Tanggap (ACT) yang memotong dana donasi secara tidak transparan untuk kepentingan operasional dan kepentingan pribadi pengelola dana seperti membeli rumah dan perabotnya, serta menerima gaji dan fasilitas lainnya dengan nilai ratusan juta rupiah (tempo.co, 2022).

Hal tersebut menimbulkan krisis kepercayaan pada pengelola dana pihak ketiga, sehingga para donator kesulitan untuk memilih proyek yang sah (Gada, 2021). Sistem penggalangan dana yang tersentralisasi menyebabkan ketergantungan pada pihak ketiga untuk evaluasi dan pengelolaan risiko, sehingga mengurangi desentralisasi dan mengingkatkan kemungkinan kesalahan atau kolusi dalam proyek. Ketergantungan pihak ke tiga juga menyebabkan biaya layanan yang tinggi (Xu, 2023). Selanjutnya, peneliti melakukan kajian literatur terkait untuk mengatasi masalah. Kajian lieteratur berupa artikel ilmiah, buku, jurnal atau website yang membahas seputar smart contract, blockchain dan implementasinya. Dengan memanfaatkan blockchain, seluruh transaksi donasi dapat dicatat secara transparan dan tidak dapat dimanipulasi.

**3.3.2 Perancangan Sistem**

Pada tahap ini akan dilakukan proses perancangan sistem berbasis blockchain Ethereum. Perancangan sistem dimulai dengan melakukan analisa kebutuhan fungsional dan kebutuhan non-fungsional, analisa pemetaan aktor dalam sistem, analisa spesifikasi sistem dan pembuatan alur sistem. Analisa kebutuhan fungsional dan non-fungsional meliputi identifikasi fitur utama yang harus dimiliki oleh sistem agar dapat beroperasi sesuai dengan tujuan yang telah di tentukan dengan melakukan observasi pada platform penggalangan online sejenis yang sudah ada.

Kebutuhan fungsional mencakup proses pengiriman Ether dari pengirim ke penerima melalui smart contract, pencatatan transaksi crowdfunding dan proses staking pada blockhain, serta mekanisme pembagian dan penarikan reward. Sementara itu, kebutuhan non-fungsional mencakup aspek teknis seperti terdesentralisasi sistem, transparansi transaksi, dan interaksi smart contract dengan user dan jaringan blockchain. Berdasarkan observasi didapatkan kebutuhan fungsional dan non-fungsional sistem dengan penyesuaian sebagai berikut:

|  |  |
| --- | --- |
|  |  |

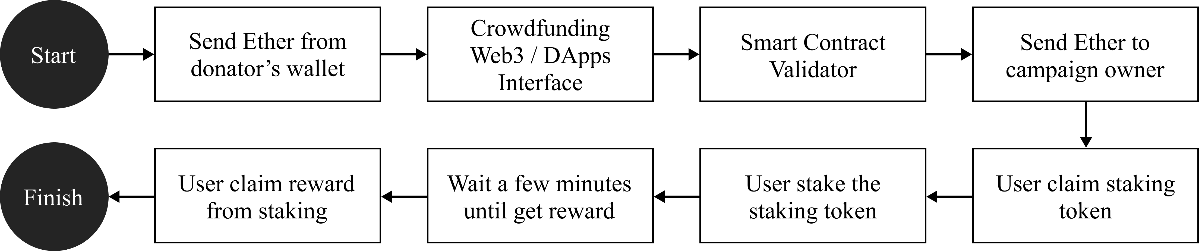
**Tabel 3**. Kebutuhan fungsional dan non-fungsional

Analisa pemetaan aktor dalam sistem mencakup identifikasi aktor yang terlibat pada penggunaan sistem crowdfunding, meliputi campaign owner, donator/backer/funder, smart contract blockhain dan web3 interface. Analisa spesifikasi sistem meliputi teknologi dan tools apa saja yang nantinya akan digunakan untuk membangun aplikasi berbasis blockchain yang terdesentralisasi, meliputi bahasa pemrograman yang digunakan untuk smart contract dan frontend, jenis blockchain dan konsensus yang digunakan, library yang digunakan untuk integrasi, dan platform Ethereum Virtual Machine (EVM) yang nantinya digunakan untuk debugging smart contract dan deploy smart contract. berikut ini adalah hasil analisa spesifikasi sistem dan tools yang digunakan.

|  |  |
| --- | --- |
| **Komponen** | **Tools dan Teknologi** |
| Blockchain | Ethereum (Sepolia Ethereum Testnet) |
| Konsensus | Ethereum (Proof of Stake) |
| Bahasa Smart Contract | Solidity |
| Provider EVM | Remix, Thirdweb |
| Library Web3 | Ethers JS, Thirdweb Provider |
| Frontend | React JS |
| Backend | Ethereum (Sepolia Ethereum Testnet) |

**Tabel 4**. Tools dan teknologi yang digunakan

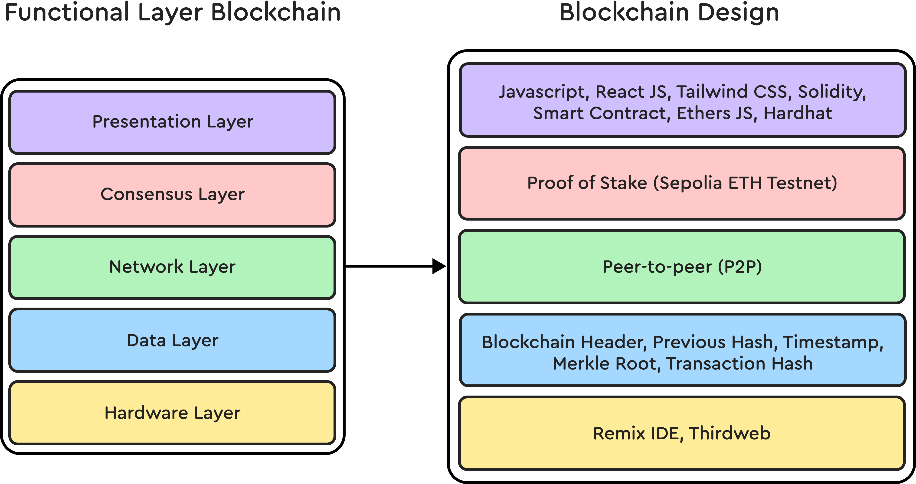
Pada sistem yang akan dibuat, pemiliki kampanye akan membuka kampanye pada aplikasi yang nantinya bisa menerima donasi dari donator. Para donator akan mengirim Ether dari wallet kripto mereka lewat interface aplikasi, Ether yang didonasikan tidak langsung dikirim ke wallet pemilik kampanye, namun akan di terima oleh smart contract untuk divalidasi dan dicatat pada blockchain. Ether yang didonasikan secara otomatis akan dikirim oleh smart contract ke pemilik kampanye jika sudah divalidasi. User bisa melakukan klaim staking token untuk menaruhkan token mereka dan nantinya akan mendapat reward. Berikut ini adalah alur transaksi pada aplikasi crowdfunding.



**Gambar 6**. Alur transaksi web3 crowdfunding

**3.3.3 Pembuatan Desain Blockchain**

Pembuatan desain blockchain dilakukan dilakuakan berdasarkan analisa spesifikasi sistem dan jaringan blockchain yang dipilih, dengan mempertimbangkan setiap layer dalam arsitektur blockchain untuk memastikan integritas yang optimal antara smart contract, konsensus dan infrastruktur jaringan. Setiap layer memiliki peran masing-masing, seperti Presentation Layer yang bertanggung jawab atas tampilan antarmuka pengguna menggunakan bahasa pembangun website, serta interaksi dengan smart contract melalui library Ethers JS. Consensus Layer mengadopsi Proof of Stake (PoS) pada jaringan blockchain Sepolia ETH untuk memvalidasi transaksi, sementara Network Layer menggunakan mode Peer-to-Peer (P2P) untuk komunikasi antar node dalam jaringan, dan Data Layer yang memastikan keamanan dan validitas data transaksi yang disimpan dengan menerapkan Transaction Hash, Hash Algorithm dan pencatatan data transaksi. Berikut ini adalah desain blockchain yang akan digunakan berdasarkan layer fungsional pada blockchain.



**Gambar 7**. Desain blockchain layer

**3.3.4 Pemodelan Smart Contract**

**3.3.5 Implementasi Smart Contract dan Web3**

**3.3.6 Pengujian**

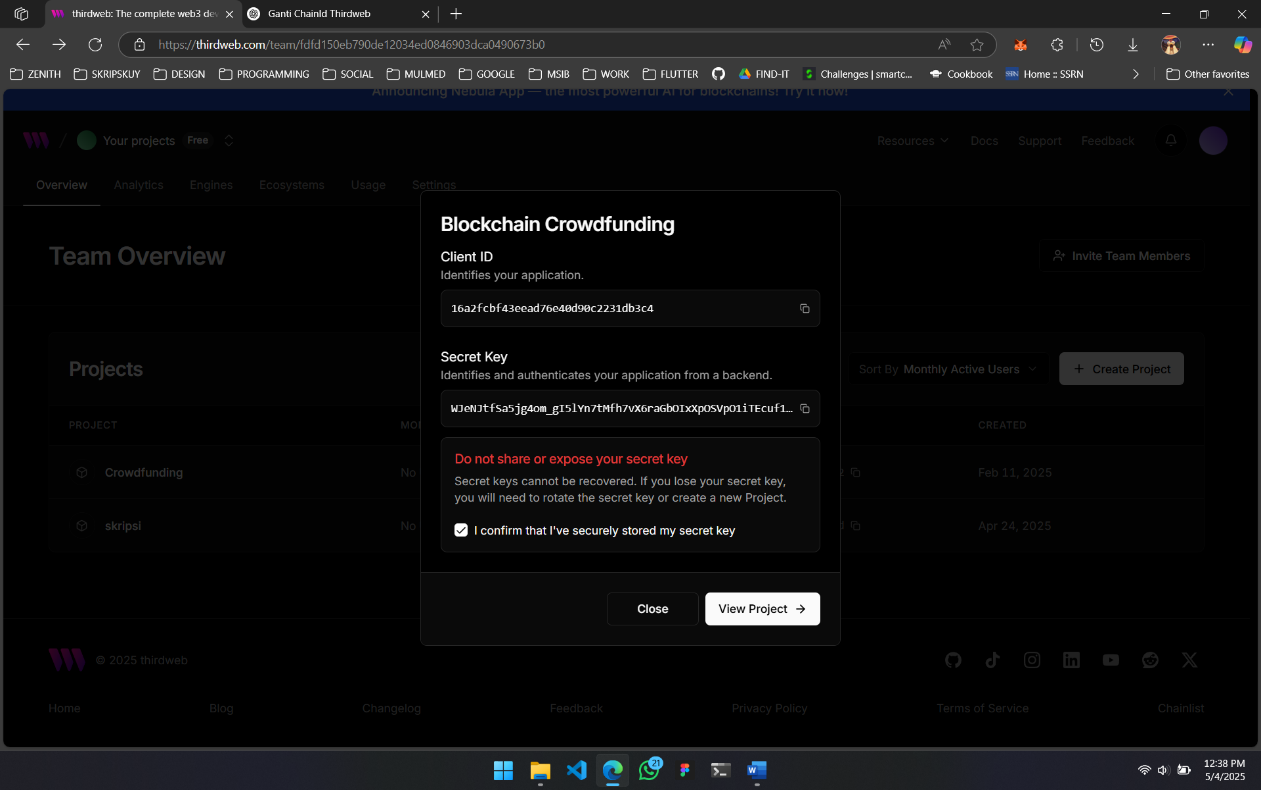
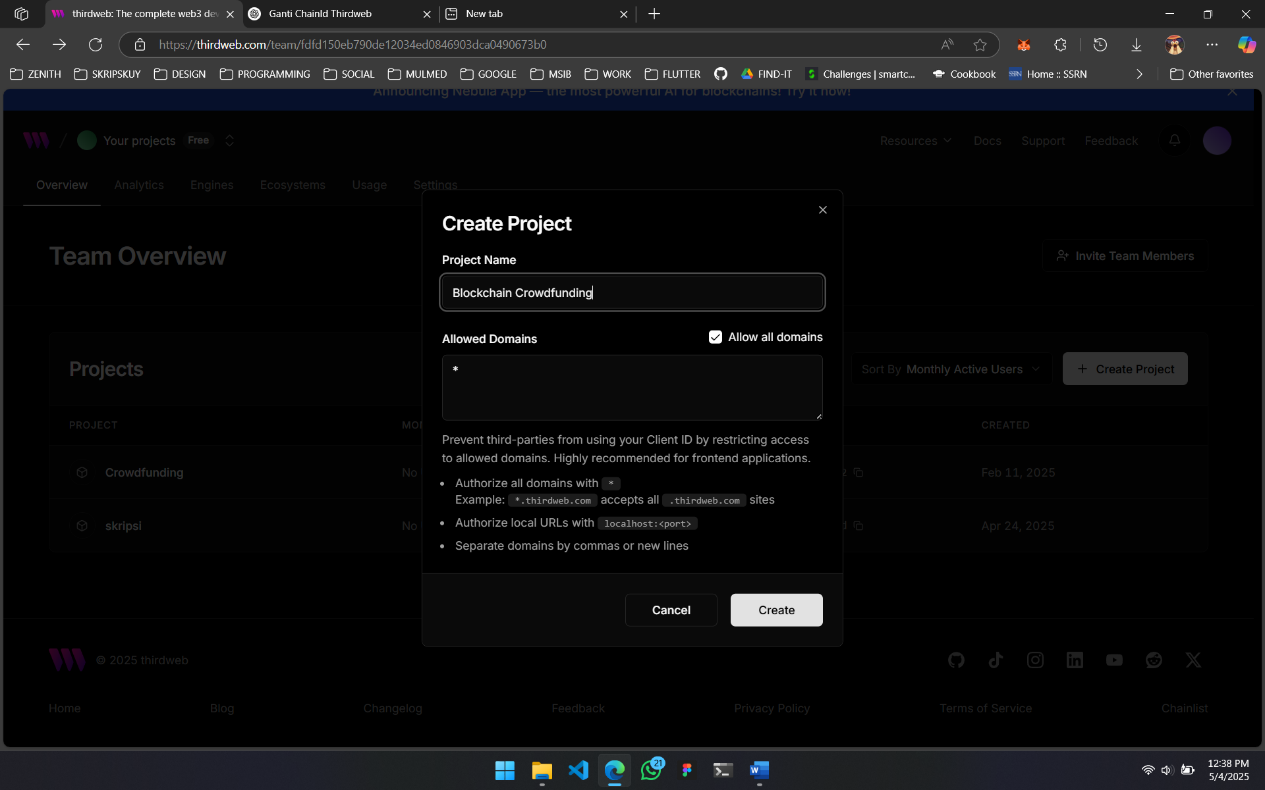
**BAB IV**

**HASIL DAN PEMBAHASAN**

**4.1 Implementasi Smart Contract dan Dapps**

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**

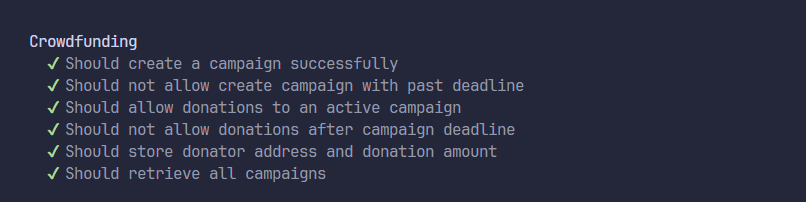
Kontrak yang sudah dideploy pada jaringan Sepolia Ethereum lewat Ethereum Virtual Machine menggunakan thirdweb akan dilakukan integrasi dengan aplikasi. Aplikasi dikembangkan menggunakan bahasa Javascript dengan menggunakan library React JS. Interface aplikasi terdiri dari beberapa halaman, antaralain; 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.

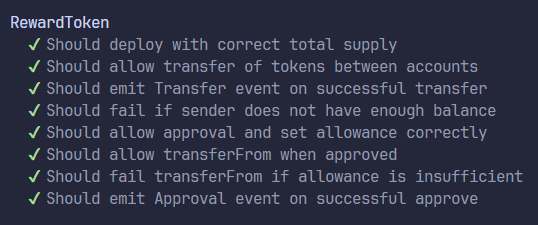
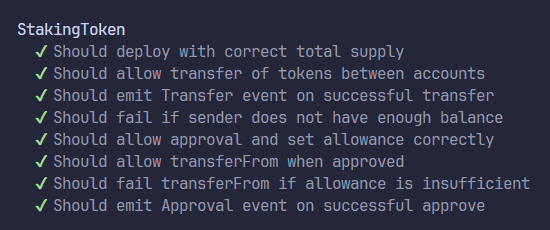


**Gambar 4.x** Hasil pengujian kontrak crowdfunding dengan Hardhat

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

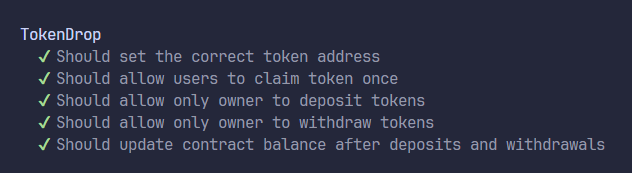
Kontrak akan diuji mengunakan Hardhat JS dengan menginputkan perintah pada terminal npx hardhat test ./test/02\_StakingToken.test.js ./test/03\_RewardToken.test.js. Kode untuk melakukan testing terdapat pada lampiran () dan di dapatkan hasil testing seperti gambar dibawah ini.

**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.

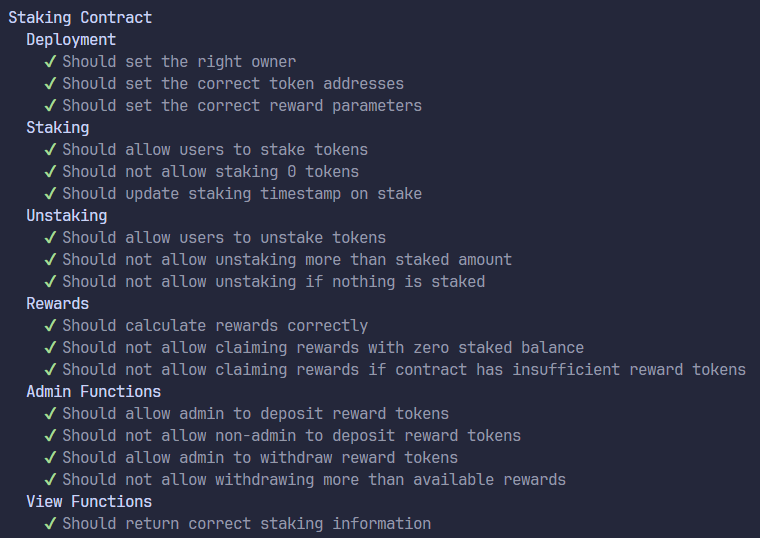


**Gambar 4.x** Hasil pengujian kontrak token drop dengan Hardhat

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.

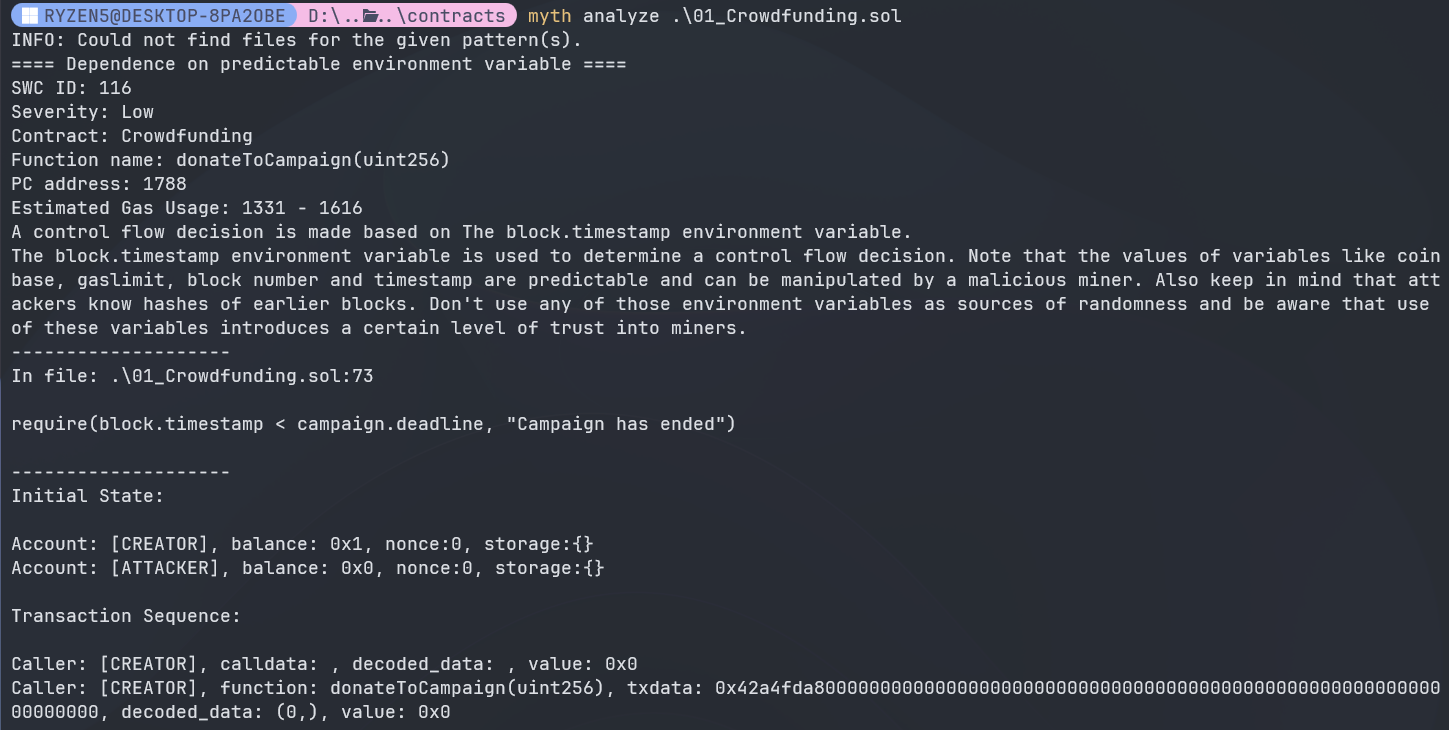


**Gambar 4.x** Hasil pengujian kontrak staking dengan Hardhat

Dari proses testing fungsional menggunakan Hardhat JS, didapatkan hasil dari pengujian 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.

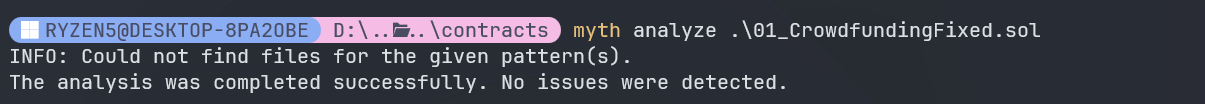
Untuk menangani kerentanan pada kode, dilakukan perbaikan kode pada smart contract pada beberapa bagian dengan menambahkan validasi pada bagian fungsi donateToCampaign. Perbaikan pada smart contract terletak pada kode dibawah.

|  |
| --- |
| ```  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

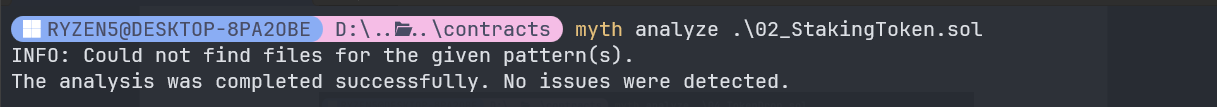
Pada fungsi donateToCampaign ditambahkan validasi yang lebih aman untuk memvalidasi deadline dari kampanye. Validasi tidak langsung menggunakan block timestamp, namun menggunakan variable isActive dan menambahkan event CampaignStatusChanged untuk mencatat perubahan pada nilai variabel isActive.

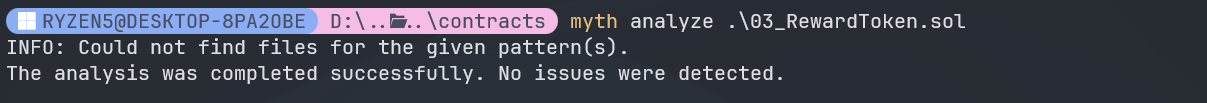
Setelah ditambahkan kode untuk validasi pada fungsi donateToCampaign, selanjutnya akan dilakukan Analisa ulang menggunakan Mythril. Testing dilakukan untuk memastikan apakah kode smart contract yang sudah di perbaiki masih memiliki celah keamanan atau tidak. Hasil pengujian ulang smart contract Crowdfunding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



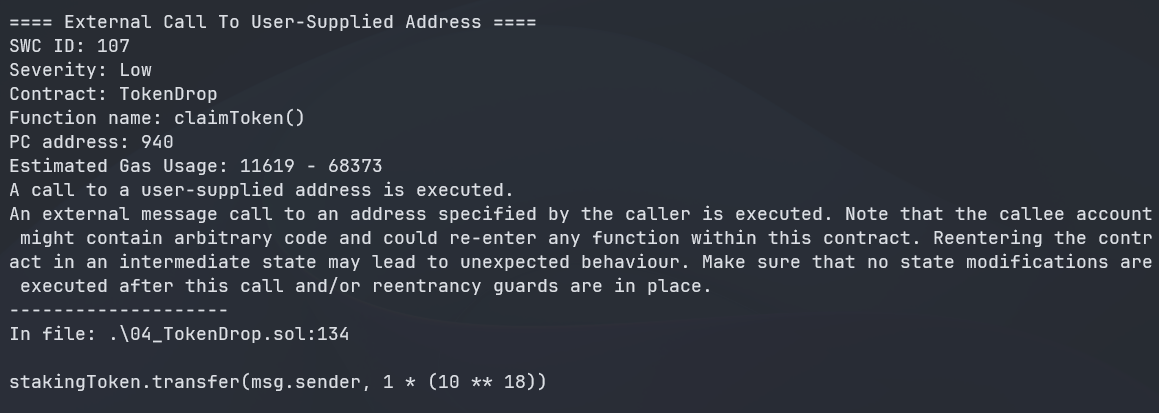
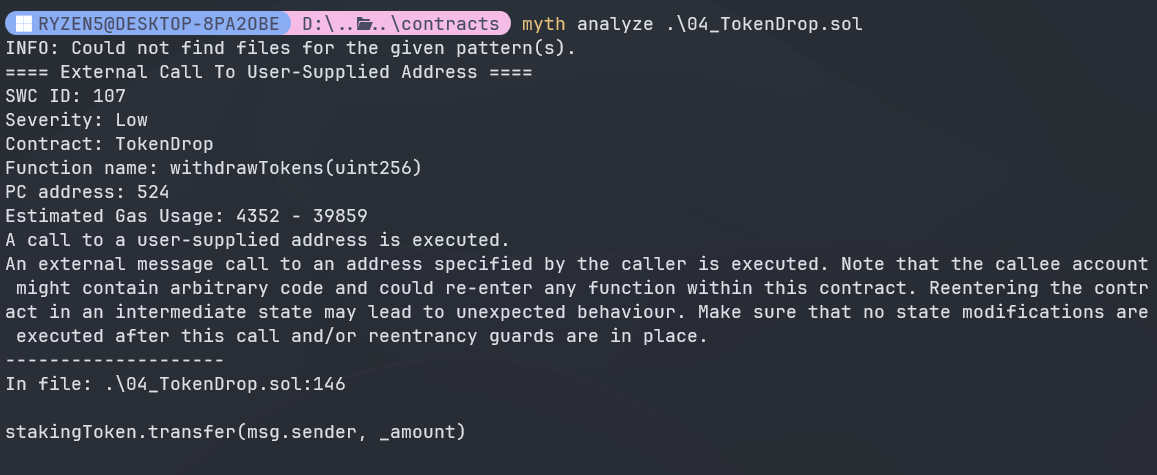
**Gambar 4.x** Hasil pengujian kembali smart contract crowdfuding

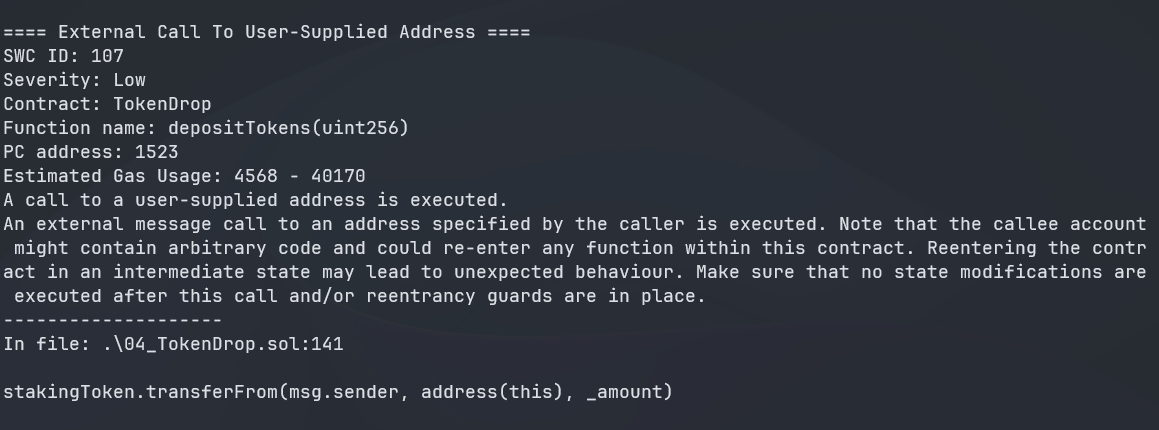
Hasil pengujian menggunakan Mythril pada kontrak Staking Token dan RewardToken adalah sebagai berikut:



**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:





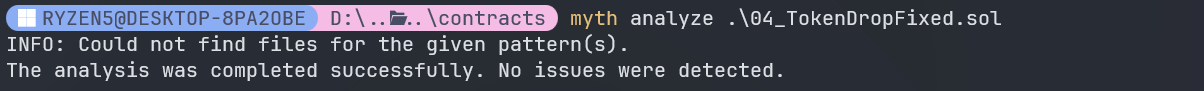
**Gambar 4.x** Hasil pengujian Mythril pada kontrak token drop

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:

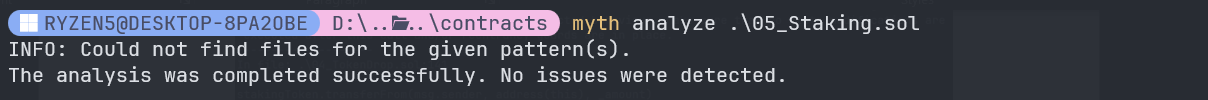
|  |
| --- |
| ```  modifier *nonReentrant*() {          require(!\_locked, "ReentrancyGuard: reentrant call");          \_locked = true;          \_;          \_locked = false;      }      function *claimToken*() external *nonReentrant* {          require(!hasClaimed[msg.sender], "You have already claimed");          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.

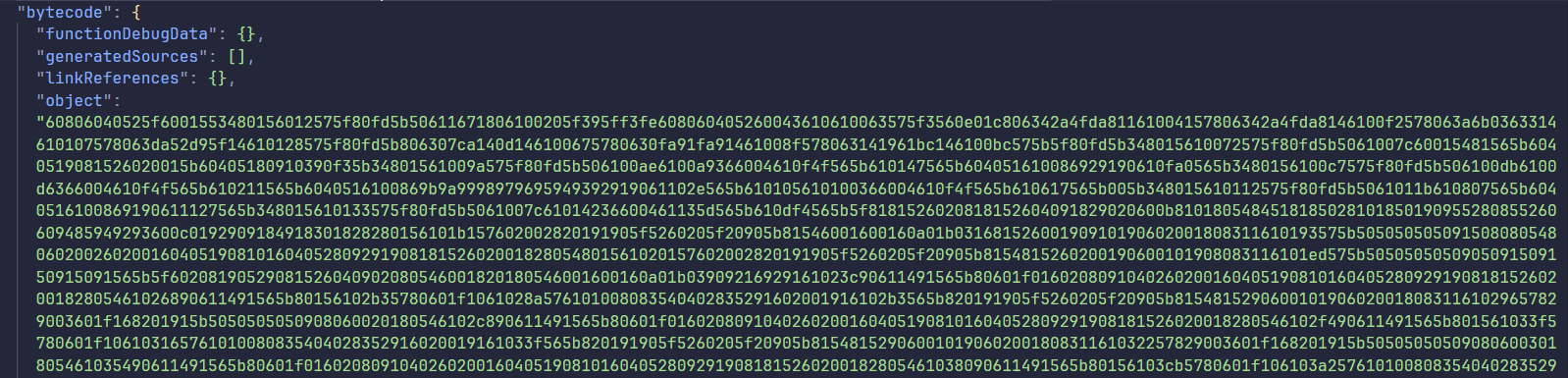
 **Gambar 4.x** Hasil pengujian kembali kontrak token drop

Hasil pengujian menggunakan Mythril pada kontrak Staking adalah sebagai berikut:

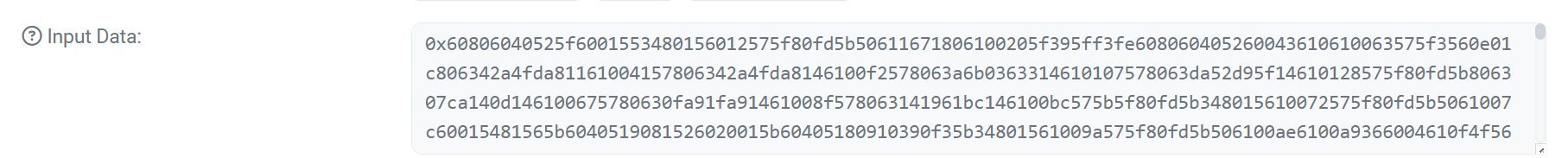


**Gambar 4.x** Hasil pengujian menggunakan Mythril pada kontrak Staking

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.

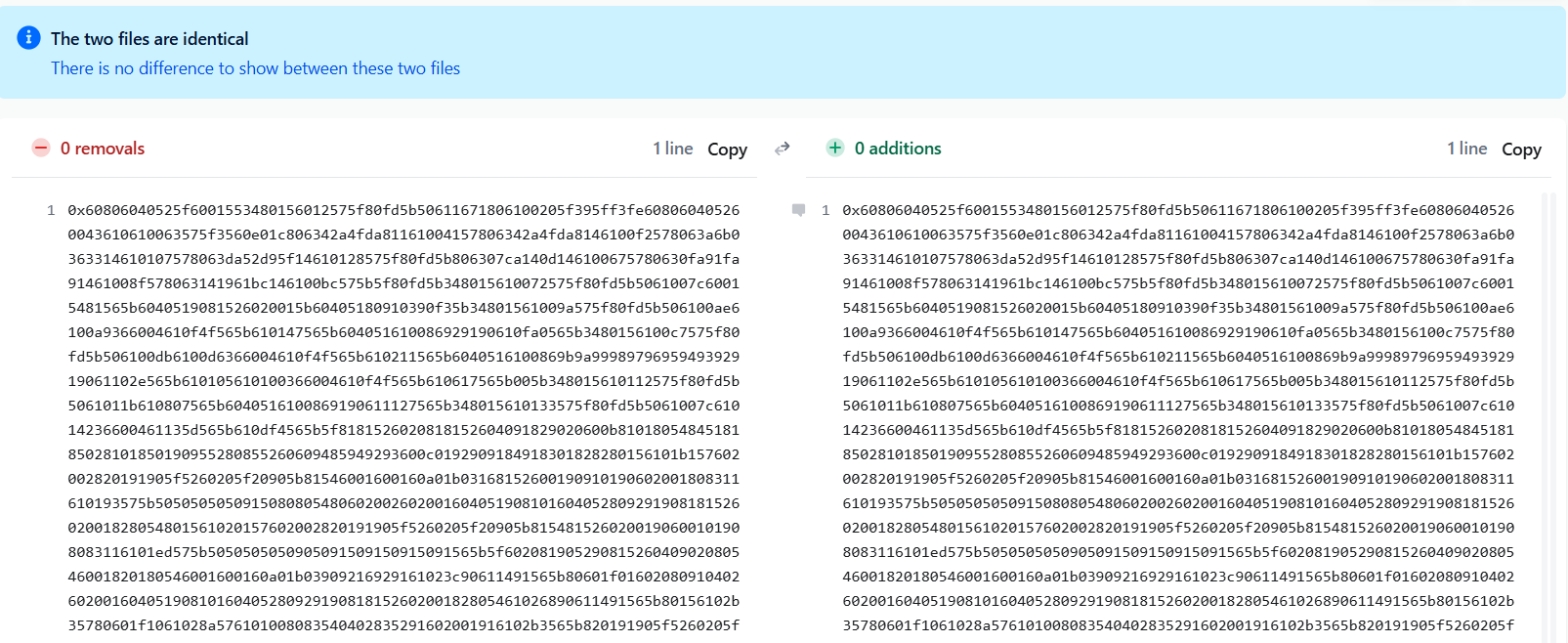


**Gambar 4.x** Bytecode hasil kompilasi smart contract



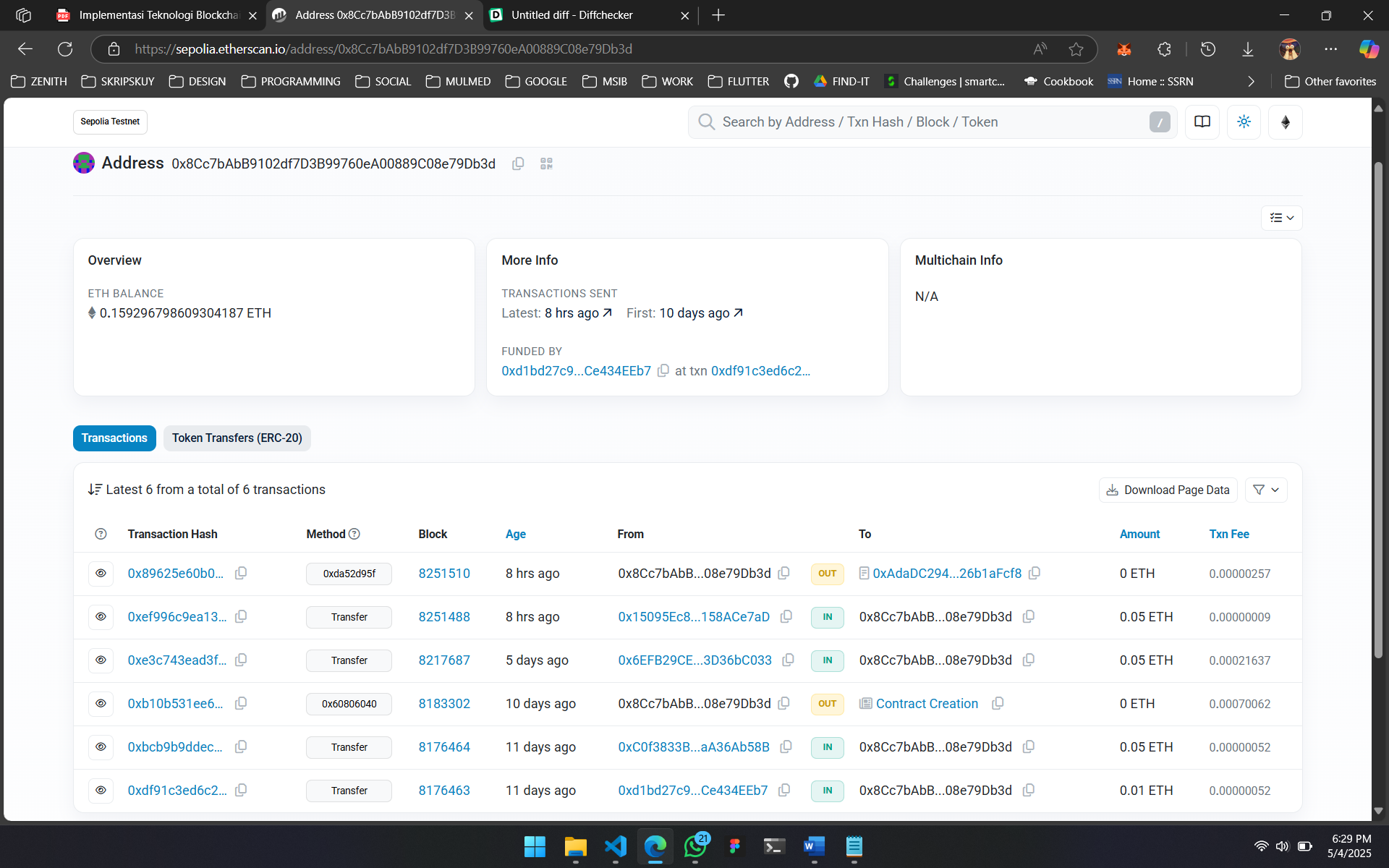
**Gambar 4.x** Bytecode smart contract pada blockchain

Ketika melakukan kompilasi solc (solidity compiler) 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.



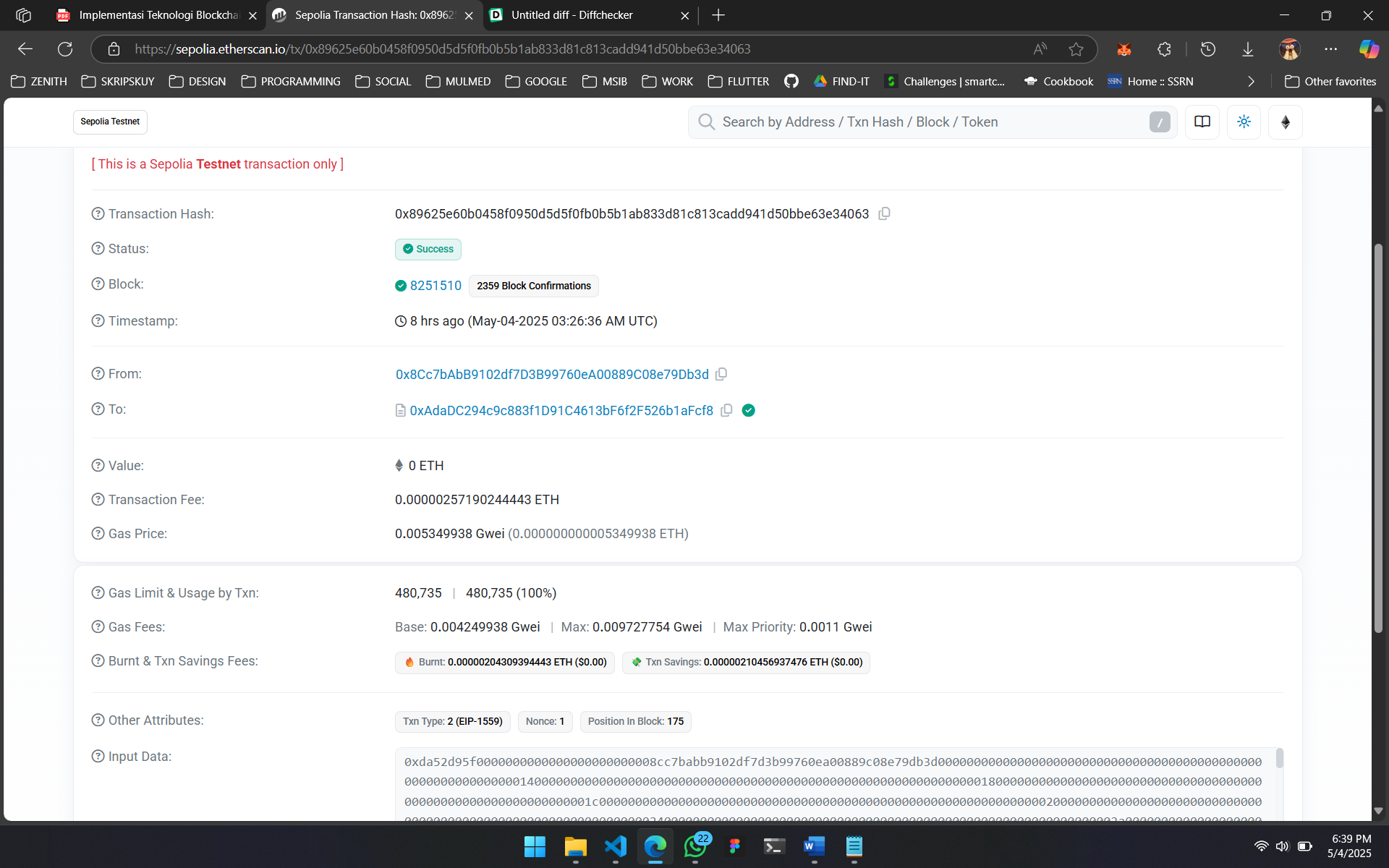
**Gambar 4.x** Hasil perbandingan bytecode

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**

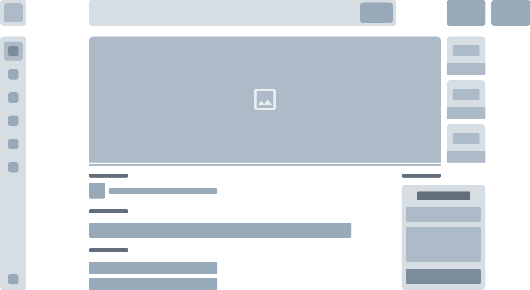
**Lampiran 1.** Wireframe dan Interface halaman dashboard



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



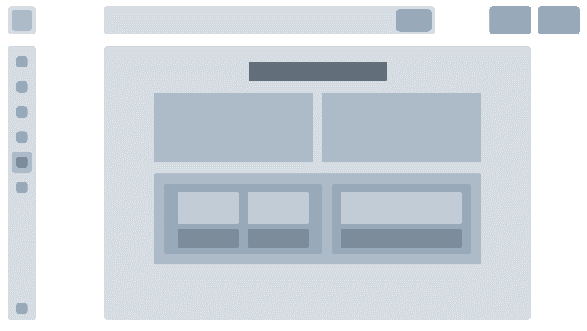
**Lampiran 3.** Wireframe dan interface halaman detail kampanye



**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);      }  } |