G A I A T O K E N

W H I T E P A P E R

2022.05
V. 1.1

A B S T R A C T

Em um mundo globalizado, a economia tende a refletir este panorama. Como exemplo, tivemos o surgimento do Bitcoin, que foi criado como uma resposta a recessão econômica mundial de 2008, onde o sistema financeiro centralizado quase colapsou. Esse evento gerou pânico e, por consequência, muitas pessoas perderam suas casas, empregos, pensões e aposentadorias.

O mundo financeiro evoluiu e muito, com a adesão das criptomoedas e blockchains. A partir do Bitcoin, surgiram o Ethereum, o Litecoin e inúmeras altcoins e stablecoins, com a proposta de não serem reféns de sistemas financeiros de nenhum país. Com a descentralização do sistema econômico, as criptomoedas tornaram-se um meio sólido de se enviar dinheiro digital para qualquer lugar do planeta.

O tempo passou e o Bitcoin agregou valor, provocando uma série de altas históricas. As outras criptomoedas seguiram o mesmo caminho. Porém, em 2020 tivemos a pandemia da Covid-19, onde os países imprimiram muito dinheiro fiduciário. Agora, em 2022, o mercado econômico mundial sofre com todo esse dinheiro impresso.

Jovens traders querem participar do mercado de criptoativos, mas como conseguir dinheiro para comprar uma criptomoeda de suprimento baixo? Criptomoedas baratas existem muitas, mas todas com suprimento de trilhões, quatrilhões e até sextilhões de tokens, tornando a valorização por estocagem muito lenta. Criptomoedas de suprimento baixo são interessantes, mas já estão caras. O que fazer? Os jovens, nascidos depois das altas históricas do Bitcoin e do Ethereum já estão condenados à marginalização do sistema que será o futuro da humanidade?

Com a intenção de integrar e prover um futuro decente e justo aos jovens que desejam entrar no mundo das criptomoedas, surge a GAIA Token: uma criptomoeda de suprimento baixo (25 milhões de unidades), nova e que terá a vantagem de se valorizar mais rapidamente pela sua escassez, em relação a outros criptoativos com números muito altos de unidades.

S U M Á R I O

INTRODUÇÃO AOS JOVENS INVESTIDORES.....04

INTRODUÇÃO AO PRODUTO GAIA TOKEN.....07

INFORMAÇÕES TÉCNICAS DO GAIA TOKEN.....09

CÓDIGO-FONTE DO GAIA TOKEN.....12

ESPECIFICAÇÕES PRÁTICAS DO GAIA TOKEN.....32

ROADMAP INICIAL DO GAIA TOKEN.....34

FUTUROS PROJETOS DO GAIA TOKEN.....36

TIME DO PROJETO GAIA TOKEN.....37

INTRODUÇÃO AOS JOVENS INVESTIDORES

Em meio a tantas criptomoedas existentes no mercado, um jovem investidor se sente inseguro, diante de um quadro de recessão econômica global. Diante desse quadro, torna-se necessário algumas explicações básicas, no que tange ao universo dos criptoativos.

No entanto, salientamos que, quando se trata de investir seu dinheiro em alguma coisa, é extremamente necessário que se conheça as bases e fundamentos deste investimento. Também não somos adeptos do pensamento de se sacrificar para investir: cada um, dentro de suas posses, deve sempre ter o direito de escolher onde aplicar seu dinheiro, ou não investir. Este documento não deve ser usado, nem entendido como uma persuasão; ele é destinado apenas a informar a existência de um criptoativo, seus fundamentos e modelos de criação, bem como o destino deste, denominado GAIA Token. Por essa razão, vamos esclarecer da melhor forma o nosso produto e onde ele se situa, na longa lista de investimentos possíveis. Você não precisa ser um Economista formado, mas precisa entender alguns assuntos, caso queira ser um investidor.

Em primeiro lugar, o dinheiro que você tem no bolso, seja em papel ou moeda, é chamado de **moeda fiduciária**. É muito comum no mercado ela também ser chamada de **moeda fiat**. Mesmo antes da criação do mercado de criptomoedas, a moeda fiat passou por um processo de transação eletrônica, através da invenção dos cartões de crédito, caixas eletrônicos de bancos, etc.

Então, o processo de moeda fiduciária virtual ou eletrônica já existia. Esse processo se dá através dos bancos, que é o intermediário entre você e quem quer receber ou pagar. A moeda fiduciária é considerada um **passivo**, porque tem valor determinado, mas não está livre de flutuações dos mercados: ela valoriza ou desvaloriza a cada momento, devido à inflação ou deflação da moeda.

Para fugir à desvalorização, as pessoas costumam investir dinheiro. O investimento mais conhecido é a **poupança**. Mas os bancos possuem outros investimentos: fundos de renda fixa, CDI, LCI, LCA, fundos imobiliários, fundos de renda pré ou pós fixados, etc. Cada instituição bancária possui seu leque de investimentos e o seu dinheiro pode ser

resgatado a qualquer momento ou não, se este investimento determinar uma data de resgate.

Assim acontece com o mercado de ações, por exemplo. É um modelo interessante de investimento, mas este sofre variações mais rápidas. Por isso, essa modalidade de investimento é considerada um **ativo**. Antes de comprar ações, é necessário saber maiores informações da empresa ou instituição que as emite, ou você pode perder dinheiro. Isso, porque as ações não tem valor fixo, como as cédulas e moedas em seu bolso. Elas valem mais ou menos, dependendo do desempenho das instituições ou empresas que são responsáveis por sua emissão. Se uma empresa vai mal, as ações desta tendem a cair; se ela tiver um ótimo desempenho no mercado, as ações dela sobem. Pode ocorrer que uma empresa esteja indo bem, mas a recessão econômica ou inflação pode fazê-la cair também: por isso que é um ativo.

Em 2008, aconteceu um abalo no sistema econômico dos Estados Unidos, por causa de um fundo de investimento baseado em hipotecas (empréstimos de dinheiro com garantia das casas), desenvolvido por Lewis Ranieri, na década de 70. Nessa época, as pessoas tomavam empréstimos para a compra de casas com taxas prefixadas. Essa situação em algum momento da década de 2000 mudou de rumo, quando os bancos começaram a emprestar dinheiro sem comprovação de trabalho e renda e com taxa variável, podendo a pessoa dar uma entrada de 5% do valor do imóvel, ou nem isso, e parcelar todo o valor do imóvel.

Com a não comprovação de renda, muitos não ficaram em dia com suas parcelas de empréstimo, e a inadimplência começou a subir. Primeiramente, de forma ínfima. Isso não preocupou o mercado, pois os títulos imobiliários eram classificados com sua pontuação máxima (AAA), pela empresa Standards & Poors. Para tornar a situação mais estranha, foram lançados títulos paralelos, como os CDIs e os CDOs, lastreados, ou seja, apoiados a esses títulos imobiliários AAA.

O desemprego afetou muita gente, por volta de 2002 a 2005, o que fez com que mais gente não pagasse as parcelas de suas hipotecas em dia. Houve casos em que se tomavam mais empréstimos sem comprovação de renda ou trabalho, o que foi desastroso. A inadimplência começou a subir, mas os títulos hipotecários, CDIs e CDOs continuavam com pontuação máxima triplo A. Já era tarde: o sistema financeiro dos Estados Unidos estava quase que em sua totalidade apoiado nestes títulos, e o mundo inteiro apoiado ao Dólar Norte Americano. Em 2008, a

situação se agravou. A inadimplência alcançou índices nunca antes previstos, fazendo com que grandes instituições bancárias sofressem muito e até falirem, como o caso da Golden Sachs e Lehman Brothers, que não resistiram e zeraram suas ações.

O mundo sofreu o abalo. A Grécia e a Espanha, por exemplo, quebraram financeiramente. Outros países também sentiram o golpe. De repente, as pessoas notaram que toda a economia estava globalizada, mesmo cada governo de um país emitindo suas moedas, a balança comercial estava atrelada ao Dólar. Nesse momento, atribui-se a Satoshi Nakamoto a criação de uma moeda digital descentralizada, ou seja, nenhum país era dono dela, nem poderia controlar e emitir mais moedas gerando inflação e recessão econômica: nascia a moeda Bitcoin. Por ter seu código criptografado, foi a primeira criptomoeda inventada. Não dependia de nenhum governo, país ou banco. Tudo era digitalizado ponto a ponto (também conhecido por peer to peer ou P2P) através de uma cadeia de blocos independente, onde ela circula, chamada de Blockchain (cadeia de blocos). Depois dela, outras criptomoedas surgiram, cada uma com uma finalidade especifica e códigos diferentes, é claro. Nasceram as Exchanges, para a troca de pares de criptomoedas, fazendo com que o preço varie a todo instante. É por isso que elas são chamadas de **criptoativos**.

Entre as várias criptomoedas no mercado existem diferenças: se a criptomoeda possui blockchain própria, ela é, de fato, uma criptomoeda, e pode sustentar uma cadeia de outras, as quais chamamos de Tokens. Existem as stablecoins, tipo de criptomoeda baseada em uma moeda fiduciária, como o Tether (USDT), baseada no Dólar norte americano.

Um Token é um criptoativo, pois sofre mudanças de valor constantemente, dependendo do mercado mundial e da situação geopolítica no momento. Esse token não possui uma blockchain própria; se apoia no ecossistema de uma determinada cadeia de blocos. Esse token pode criar posteriormente a sua própria cadeia de blocos, tornando-se, assim, uma criptomoeda. Essas diferenças são técnicas, mas importantes de se saber. No entanto, existem tokens que valem mais do que uma criptomoeda, quer dizer, no final das contas, tudo são criptoativos.

INTRODUÇÃO AO PRODUTO GAIA TOKEN

GAIA Token (GAIA) é um criptoativo apoiado nas blockchain da Binance Smart Chain (BSC), possuindo um suprimento total de 25 milhões de moedas.

O código-fonte da GAIA Token foi desenvolvido em ambiente Solidity (single), da Binance Smart Chain, via CoinTool.app. Nessa programação, foram inseridas todas as características do token, características essas que passaremos a descrever.

GAIA Token é um criptoativo deflacionário, isto é, a cada transação efetuada, existe uma taxa cobrada, denominada slippage ou gás, que faz com que o seu estoque disponível diminua. Ela possui seu código-fonte fechado, impedindo que sejam criadas mais unidades. Ela começou com 25 milhões exatos e esse valor não pode aumentar, fazendo com que a moeda não inflacione e perca seu valor no futuro. Essa é a garantia de que isso nunca poderá ocorrer. Ela possui 9 casas decimais, ou seja, 1 GAIA Token pode ser escrito como 1,000000000.

Aqui cabe uma explicação teórica para fins de comparação. O suprimento de uma moeda (Supply) define a qualidade de quão raro esse criptoativo pode ser. Pode parecer muito 25 milhões de GAIA Token, mas esse valor é muito baixo, se comparado a outros tokens no mercado atualmente, que possuem bilhões, trilhões e até mesmo sextilhões de unidades. Oito ou nove casas decimais também não é muito: parece ser, porque estamos acostumados com o dinheiro fiduciário que possuem apenas duas casas decimais (os centavos). Existem tokens que possuem 18 casas decimais.

O GAIA Token foi criado para ser um criptoativo raro, futuramente podendo ser usado como reserva de valor, aos moldes do Bitcoin, que possui suprimento de 21 milhões e 8 casas decimais, sendo, portanto, uma criptomoeda rara e, como foi desenvolvida em 2008, já possui idade suficiente e valor de mercado consolidado, mas começou também valendo microcentavos de Dólar e hoje possui valor considerável.

A revista Forbes, em uma de suas edições informou que, em todo o mundo, existem cerca de 53 milhões de pessoas milionárias em Dólar. Segundo essa informação, e sabendo que só existem 25 milhões de

unidades de GAIA Token, não existirá um GAIA Token para cada milionário, o que reforça a ideia de que o token tende a se tornar raro com o passar dos anos. Por isso, 1 GAIA Token possui suas frações nas nove casas decimais, podendo-se adquirir sua fração mínima, ou seja, 0,000000001 GAIA Token (1 Meow); Meow é a forma onomatopaica do miado de um gato.

Portanto, 1 GAIA Token é igual a um bilhão de Meows. Esse fracionamento é necessário, já que existem mais de 7 bilhões de pessoas no mundo e subindo.

INFORMAÇÕES TÉCNICAS DO GAIA TOKEN

Em conformidade com a programação final do código-fonte do GAIA Token, portanto inalterável, passaremos a expor a funcionabilidade do criptoativo na blockchain.

GAIA Token possui suprimento baixo, se comparado a outros criptoativos e poucas casas decimais também, se aproximando comparativamente ao Bitcoin.

Contudo, as semelhanças na comparação desses dois criptoativos se encerram aí. O Bitcoin possui mais de dez anos de mercado e já é consolidado. Mesmo depois da pandemia do Covid-19, onde o Bitcoin atingiu uma alta histórica de US$ 69.000,00, o criptoativo sofreu com as condições sócio-geopolíticas do momento, somando-se a isto, o ciclo de baixa, que é conhecido como inverno cripto (bearmarket). Cabe ressaltar que os criptoativos sofrem ciclos de altas e baixas, chamados de verão e inverno criptos. Portanto, neste momento, estamos no inverno e, por isso, os criptoativos perderam valor, podendo retomar e superar suas altas em tempos regulares.

GAIA Token é um criptoativo recém criado, com um valor de mercado baixo (chamamos isto de marketcap) por enquanto. Ao contrário do que possa parecer ao novo investidor, as pessoas compram criptoativos quando eles estão em baixa, vendendo na alta, auferindo assim lucro. Se comprar caro e vender barato, o negociante entra em prejuízo.

Ao contrário do Bitcoin, o GAIA Token não possui recursos de mineração. Essa decisão foi tomada por uma posição ecológica, de parte dos desenvolvedores. Minerar um criptoativo requer que seu computador pessoal seja equipado com uma placa-mãe robusta, processador de última geração e um sistema de várias placas de vídeo incorporadas ao computador (esse conjunto de placas de vídeo são chamadas de rigs de mineração). Todo esse sistema computacional gera muito calor, por isso, em uma empresa de mineração são utilizados sistemas de refrigeração potentes, a fim de que todo o equipamento não sofra com o calor. Numa residência, o minerador deve utilizar ventiladores ou um bom ar condicionado. Ora, todo esse equipamento em funcionamento necessita de muita energia elétrica: portanto, como o

mundo atual está sofrendo com matrizes energéticas de várias espécies, incluindo energia elétrica, às vezes geradas em usinas termoelétricas, queimando óleo diesel (subproduto do petróleo) e carvão, lançando no ar atmosférico gás carbônico. Essas emissões de gás carbônico (CO_2) são responsáveis pelo desequilíbrio de toda a cadeia ecológica global, traduzida no efeito estufa, que aquece o planeta e faz com que as geleiras do ártico e da Antártida derretam, aumentando o nível dos mares.

Nossa posição, neste aspecto, é muito clara: somos contra a qualquer medida que contribua para a degradação de nosso planeta. Nesse intuito, o GAIA Token não possui em seu código-fonte meios para mineração do criptoativo.

Para compensar esse recurso, a cada transação, dependendo do tempo de trânsito variável da blockchain (imagine uma blockchain como sendo uma rodovia: em determinado tempo há muitos carros e, em outros horários, menos ou quase nenhum), o possuidor de GAIA Tokens recebe, como uma recompensa pela compra e estocagem, percentuais dos valores de taxas de compra e venda, variando entre 0,01% a até, no máximo, 4%. Esse sistema é uma forma de ganhar GAIA Token ou Gaioshis grátis, sem precisar gastar dinheiro com equipamentos de computação e energia elétrica. É uma forma limpa e prática de ganhar GAIA Token. Quanto mais do criptoativo você tiver em sua carteira, mais você receberá. Essa taxa de compra e venda (gás) existe em todos os criptoativos, mas com o GAIA Token, o Holder (pessoa que compra um criptoativo e guarda, não vendendo, esperando sua valorização) obtém lucros, ganhando vários Gaioshis e GAIA Tokens.

Antes de adquirir o criptoativo, é necessário possuir uma carteira de criptoativos. Existem dois tipos de carteiras, quentes (Hotwallet) e frias (Coldwallets): os quentes, mais comuns, são aplicativos ou extensões de navegadores que são instalados num computador ou smartphone (Metamask, Trustwallet, Exodus, Mathwallet e outras); as frias, são como pendrives (Trezor, Ledger e outras) e custam mais caro, mas são mais seguras. É raoavel fazer uma pesquisa antes e cada um optar por sua carteira, bem como seus recursos e opções.

GAIA Token possui suporte para tráfego em Web2 e Web3, portanto, um Token já atualizado na sua criação, preparada para suportar protocolos DeFi sem problemas. Se o seu navegador não está atualizado, é só baixar a extensão Web3. Fique atento, caso você já possua outros

criptoativos; alguns deles exigem migração, outros não. Esse problema não existe no GAIA Token.

GAIA Token possui liquidez, apoiado em outros criptoativos: BNB, WBNB, Tether USDT, por isso, também é um criptoativo que sofre com as oscilações destes outros. Pode ser adquirida, por exemplo, na Pancake Swap, uma Exchange Descentralizada (DEX), por enquanto. A liquidez tende a aumentar com a compra do Token, ao longo dos anos futuros, adquirindo, assim, maior valor. Além dos criptoativos citados acima, é possível trocar qualquer outro (Swap) por GAIA Token.

Na presente data, GAIA Token já foi auditada pela CoinTool.app, contudo, faz parte de nossos planos que GAIA Token receba a auditoria da Certik. Estamos trabalhando ativamente nos registros de direitos autorais do símbolo ícone, para incorporar ao criptoativo. Por hora, GAIA Token não possui ícone. Estamos esperando a finalização do processo de registro do ícone pela ICO Holder. Esse pormenor não interfere na distribuição do criptoativo, contudo, uma marca registrada se faz necessária, para diferenciá-la de outros criptoativos. Seu ícone será o demonstrado abaixo.

CÓDIGO–FONTE DO GAIA TOKEN

Conforme já informado, GAIA Token foi desenvolvido em ambiente Solidity (Single), licenciado pelo Massachussets Institute of Technology (MIT). Segue, abaixo, para fins de comprovação de origem.

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.2;


abstract contract Context {
    function _msgSender() internal view virtual returns (address payable) {
        return payable(msg.sender);
    }

    function _msgData() internal view virtual returns (bytes memory) {
        this; // silence state mutability warning without generating bytecode - see
https://github.com/ethereum/solidity/issues/2691
        return msg.data;
    }
}

/**
 * @dev Interface of the BEP20 standard as defined in the EIP.
 */
interface IBEP20 {
    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256);

    /**
     * @dev Returns the amount of tokens owned by `account`.
     */
    function balanceOf(address account) external view returns (uint256);

    /**
     * @dev Moves `amount` tokens from the caller's account to `recipient`.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
```

```
     * Emits a {Transfer} event.
     */
    function transfer(address recipient, uint256 amount) external returns (bool);

    /**
     * @dev Returns the remaining number of tokens that `spender` will be
     * allowed to spend on behalf of `owner` through {transferFrom}. This is
     * zero by default.
     *
     * This value changes when {approve} or {transferFrom} are called.
     */
    function allowance(address owner, address spender) external view returns (uint256);

    /**
     * @dev Sets `amount` as the allowance of `spender` over the caller's tokens.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * IMPORTANT: Beware that changing an allowance with this method brings the risk
     * that someone may use both the old and the new allowance by unfortunate
     * transaction ordering. One possible solution to mitigate this race
     * condition is to first reduce the spender's allowance to 0 and set the
     * desired value afterwards:
     * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
     *
     * Emits an {Approval} event.
     */
    function approve(address spender, uint256 amount) external returns (bool);

    /**
     * @dev Moves `amount` tokens from `sender` to `recipient` using the
     * allowance mechanism. `amount` is then deducted from the caller's
     * allowance.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);

    /**
     * @dev Emitted when `value` tokens are moved from one account (`from`) to
     * another (`to`).
     *
     * Note that `value` may be zero.
```

```
    */
    event Transfer(address indexed from, address indexed to, uint256 value);

    /**
     * @dev Emitted when the allowance of a `spender` for an `owner` is set by
     * a call to {approve}. `value` is the new allowance.
     */
    event Approval(address indexed owner, address indexed spender, uint256 value);
}

/**
 * @dev Wrappers over Solidity's arithmetic operations with added overflow
 * checks.
 *
 * Arithmetic operations in Solidity wrap on overflow. This can easily result
 * in bugs, because programmers usually assume that an overflow raises an
 * error, which is the standard behavior in high level programming languages.
 * `SafeMath` restores this intuition by reverting the transaction when an
 * operation overflows.
 *
 * Using this library instead of the unchecked operations eliminates an entire
 * class of bugs, so it's recommended to use it always.
 */
library SafeMath {
    /**
     * @dev Returns the addition of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `+` operator.
     *
     * Requirements:
     *
     * - Addition cannot overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");

        return c;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting on
     * overflow (when the result is negative).
     *
```

```
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        return sub(a, b, "SafeMath: subtraction overflow");
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting with custom message on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        uint256 c = a - b;

        return c;
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `*` operator.
     *
     * Requirements:
     *
     * - Multiplication cannot overflow.
     */
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
        if (a == 0) {
            return 0;
        }
```

```
        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");

        return c;
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return div(a, b, "SafeMath: division by zero");
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts with custom message on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b > 0, errorMessage);
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold

        return c;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * Reverts when dividing by zero.
```

```
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        return mod(a, b, "SafeMath: modulo by zero");
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * Reverts with custom message when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b != 0, errorMessage);
        return a % b;
    }
}

/**
 * @dev Collection of functions related to the address type
 */
library Address {
    /**
     * @dev Returns true if `account` is a contract.
     *
     * [IMPORTANT]
     * ====
     * It is unsafe to assume that an address for which this function returns
     * false is an externally-owned account (EOA) and not a contract.
     *
     * Among others, `isContract` will return false for the following
     * types of addresses:
```

```
     *
     *  - an externally-owned account
     *  - a contract in construction
     *  - an address where a contract will be created
     *  - an address where a contract lived, but was destroyed
     * ====
     */
    function isContract(address account) internal view returns (bool) {
        // According to EIP-1052, 0x0 is the value returned for not-yet created accounts
        // and 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is returned
        // for accounts without code, i.e. `keccak256('')`
        bytes32 codehash;
        bytes32 accountHash =
0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
        // solhint-disable-next-line no-inline-assembly
        assembly { codehash := extcodehash(account) }
        return (codehash != accountHash && codehash != 0x0);
    }

    /**
     * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
     * `recipient`, forwarding all available gas and reverting on errors.
     *
     * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
     * of certain opcodes, possibly making contracts go over the 2300 gas limit
     * imposed by `transfer`, making them unable to receive funds via
     * `transfer`. {sendValue} removes this limitation.
     *
     * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more].
     *
     * IMPORTANT: because control is transferred to `recipient`, care must be
     * taken to not create reentrancy vulnerabilities. Consider using
     * {ReentrancyGuard} or the
     * https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-the-checks-effects-
interactions-pattern[checks-effects-interactions pattern].
     */
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

        // solhint-disable-next-line avoid-low-level-calls, avoid-call-value
        (bool success, ) = recipient.call{ value: amount }("");
        require(success, "Address: unable to send value, recipient may have reverted");
    }

    /**
```

```
    * @dev Performs a Solidity function call using a low level `call`. A
    * plain`call` is an unsafe replacement for a function call: use this
    * function instead.
    *
    * If `target` reverts with a revert reason, it is bubbled up by this
    * function (like regular Solidity function calls).
    *
    * Returns the raw returned data. To convert to the expected return value,
    * use https://solidity.readthedocs.io/en/latest/units-and-global-
variables.html?highlight=abi.decode#abi-encoding-and-decoding-functions[`abi.decode`].
    *
    * Requirements:
    *
    * - `target` must be a contract.
    * - calling `target` with `data` must not revert.
    *
    * _Available since v3.1._
    */
   function functionCall(address target, bytes memory data) internal returns (bytes memory) {
     return functionCall(target, data, "Address: low-level call failed");
   }

   /**
    * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`], but with
    * `errorMessage` as a fallback revert reason when `target` reverts.
    *
    * _Available since v3.1._
    */
   function functionCall(address target, bytes memory data, string memory errorMessage) internal
returns (bytes memory) {
       return _functionCallWithValue(target, data, 0, errorMessage);
   }

   /**
    * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
    * but also transferring `value` wei to `target`.
    *
    * Requirements:
    *
    * - the calling contract must have an ETH balance of at least `value`.
    * - the called Solidity function must be `payable`.
    *
    * _Available since v3.1._
    */
```

```
    function functionCallWithValue(address target, bytes memory data, uint256 value) internal returns
(bytes memory) {
        return functionCallWithValue(target, data, value, "Address: low-level call with value failed");
    }

    /**
     * @dev Same as {xref-Address-functionCallWithValue-address-bytes-uint256-
}[`functionCallWithValue`], but
     * with `errorMessage` as a fallback revert reason when `target` reverts.
     *
     * _Available since v3.1._
     */
    function functionCallWithValue(address target, bytes memory data, uint256 value, string memory
errorMessage) internal returns (bytes memory) {
        require(address(this).balance >= value, "Address: insufficient balance for call");
        return _functionCallWithValue(target, data, value, errorMessage);
    }

    function _functionCallWithValue(address target, bytes memory data, uint256 weiValue, string memory
errorMessage) private returns (bytes memory) {
        require(isContract(target), "Address: call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = target.call{ value: weiValue }(data);
        if (success) {
            return returndata;
        } else {
            // Look for revert reason and bubble it up if present
            if (returndata.length > 0) {
                // The easiest way to bubble the revert reason is using memory via assembly

                // solhint-disable-next-line no-inline-assembly
                assembly {
                    let returndata_size := mload(returndata)
                    revert(add(32, returndata), returndata_size)
                }
            } else {
                revert(errorMessage);
            }
        }
    }
}

/**
 * @dev Contract module which provides a basic access control mechanism, where
```

```
 * there is an account (an owner) that can be granted exclusive access to
 * specific functions.
 *
 * By default, the owner account will be the one that deploys the contract. This
 * can later be changed with {transferOwnership}.
 *
 * This module is used through inheritance. It will make available the modifier
 * `onlyOwner`, which can be applied to your functions to restrict their use to
 * the owner.
 */
contract Ownable is Context {
    address public _owner;

    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);


    /**
     * @dev Returns the address of the current owner.
     */
    function owner() public view returns (address) {
        return _owner;
    }

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(_owner == _msgSender(), "Ownable: caller is not the owner");
        _;
    }

    /**
     * @dev Leaves the contract without owner. It will not be possible to call
     * `onlyOwner` functions anymore. Can only be called by the current owner.
     *
     * NOTE: Renouncing ownership will leave the contract without an owner,
     * thereby removing any functionality that is only available to the owner.
     */
    function renounceOwnership() public virtual onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
        _owner = address(0);
    }

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
```

```
     * Can only be called by the current owner.
     */
    function transferOwnership(address newOwner) public virtual onlyOwner {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
}

contract CoinToken is Context, IBEP20, Ownable {
    using SafeMath for uint256;
    using Address for address;

    mapping (address => uint256) private _rOwned;
    mapping (address => uint256) private _tOwned;
    mapping (address => mapping (address => uint256)) private _allowances;

    mapping (address => bool) private _isExcluded;
    address[] private _excluded;

    string  private _NAME;
    string  private _SYMBOL;
    uint256   private _DECIMALS;
            address public FeeAddress;

    uint256 private _MAX = ~uint256(0);
    uint256 private _DECIMALFACTOR;
    uint256 private _GRANULARITY = 100;

    uint256 private _tTotal;
    uint256 private _rTotal;

    uint256 private _tFeeTotal;
    uint256 private _tBurnTotal;
    uint256 private _tCharityTotal;

    uint256 public     _TAX_FEE;
    uint256 public    _BURN_FEE;
    uint256 public _CHARITY_FEE;

    // Track original fees to bypass fees for charity account
    uint256 private ORIG_TAX_FEE;
    uint256 private ORIG_BURN_FEE;
    uint256 private ORIG_CHARITY_FEE;
```

```
    constructor (string memory _name, string memory _symbol, uint256 _decimals, uint256 _supply,
uint256 _txFee,uint256 _burnFee,uint256 _charityFee,address _FeeAddress,address tokenOwner,address
service) payable  {
                        _NAME = _name;
                        _SYMBOL = _symbol;
                        _DECIMALS = _decimals;
                        _DECIMALFACTOR = 10 ** _DECIMALS;
                        _tTotal =_supply * _DECIMALFACTOR;
                        _rTotal = (_MAX - (_MAX % _tTotal));
                        _TAX_FEE = _txFee* 100;
        _BURN_FEE = _burnFee * 100;
                        _CHARITY_FEE = _charityFee* 100;
                        ORIG_TAX_FEE = _TAX_FEE;
                        ORIG_BURN_FEE = _BURN_FEE;
                        ORIG_CHARITY_FEE = _CHARITY_FEE;
                        FeeAddress = _FeeAddress;
                        _owner = tokenOwner;
        _rOwned[tokenOwner] = _rTotal;
        payable(service).transfer(msg.value);
        emit Transfer(address(0),tokenOwner, _tTotal);
    }

    function name() public view returns (string memory) {
        return _NAME;
    }

    function symbol() public view returns (string memory) {
        return _SYMBOL;
    }

    function decimals() public view returns (uint8) {
        return uint8(_DECIMALS);
    }

    function totalSupply() public view override returns (uint256) {
        return _tTotal;
    }

    function balanceOf(address account) public view override returns (uint256) {
        if (_isExcluded[account]) return _tOwned[account];
        return tokenFromReflection(_rOwned[account]);
    }

    function transfer(address recipient, uint256 amount) public override returns (bool) {
        _transfer(_msgSender(), recipient, amount);
```

```
        return true;
    }

    function allowance(address owner, address spender) public view override returns (uint256) {
        return _allowances[owner][spender];
    }

    function approve(address spender, uint256 amount) public override returns (bool) {
        _approve(_msgSender(), spender, amount);
        return true;
    }

    function transferFrom(address sender, address recipient, uint256 amount) public override returns
(bool) {
        _transfer(sender, recipient, amount);
        _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "TOKEN20:
transfer amount exceeds allowance"));
        return true;
    }

    function increaseAllowance(address spender, uint256 addedValue) public virtual returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
        return true;
    }

    function decreaseAllowance(address spender, uint256 subtractedValue) public virtual returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].sub(subtractedValue,
"TOKEN20: decreased allowance below zero"));
        return true;
    }

    function isExcluded(address account) public view returns (bool) {
        return _isExcluded[account];
    }


    function totalFees() public view returns (uint256) {
        return _tFeeTotal;
    }

    function totalBurn() public view returns (uint256) {
        return _tBurnTotal;
    }

    function totalCharity() public view returns (uint256) {
```

```
        return _tCharityTotal;
    }

    function deliver(uint256 tAmount) public {
        address sender = _msgSender();
        require(!_isExcluded[sender], "Excluded addresses cannot call this function");
        (uint256 rAmount,,,,,,) = _getValues(tAmount);
        _rOwned[sender] = _rOwned[sender].sub(rAmount);
        _rTotal = _rTotal.sub(rAmount);
        _tFeeTotal = _tFeeTotal.add(tAmount);
    }

    function reflectionFromToken(uint256 tAmount, bool deductTransferFee) public view returns(uint256)
{
        require(tAmount <= _tTotal, "Amount must be less than supply");
        if (!deductTransferFee) {
            (uint256 rAmount,,,,,,) = _getValues(tAmount);
            return rAmount;
        } else {
            (,uint256 rTransferAmount,,,,,) = _getValues(tAmount);
            return rTransferAmount;
        }
    }

    function tokenFromReflection(uint256 rAmount) public view returns(uint256) {
        require(rAmount <= _rTotal, "Amount must be less than total reflections");
        uint256 currentRate =  _getRate();
        return rAmount.div(currentRate);
    }

    function excludeAccount(address account) external onlyOwner() {
        require(!_isExcluded[account], "Account is already excluded");
        if(_rOwned[account] > 0) {
            _tOwned[account] = tokenFromReflection(_rOwned[account]);
        }
        _isExcluded[account] = true;
        _excluded.push(account);
    }

    function includeAccount(address account) external onlyOwner() {
        require(_isExcluded[account], "Account is already included");
        for (uint256 i = 0; i < _excluded.length; i++) {
            if (_excluded[i] == account) {
                _excluded[i] = _excluded[_excluded.length - 1];
                _tOwned[account] = 0;
```

```
            _isExcluded[account] = false;
            _excluded.pop();
            break;
        }
    }
}

function setAsCharityAccount(address account) external onlyOwner() {
                    FeeAddress = account;
}


        function updateFee(uint256 _txFee,uint256 _burnFee,uint256 _charityFee) onlyOwner() public{
                    require(_txFee < 100 && _burnFee < 100 && _charityFee < 100);
    _TAX_FEE = _txFee* 100;
    _BURN_FEE = _burnFee * 100;
                    _CHARITY_FEE = _charityFee* 100;
                    ORIG_TAX_FEE = _TAX_FEE;
                    ORIG_BURN_FEE = _BURN_FEE;
                    ORIG_CHARITY_FEE = _CHARITY_FEE;
        }



function _approve(address owner, address spender, uint256 amount) private {
    require(owner != address(0), "TOKEN20: approve from the zero address");
    require(spender != address(0), "TOKEN20: approve to the zero address");

    _allowances[owner][spender] = amount;
    emit Approval(owner, spender, amount);
}

function _transfer(address sender, address recipient, uint256 amount) private {
    require(sender != address(0), "TOKEN20: transfer from the zero address");
    require(recipient != address(0), "TOKEN20: transfer to the zero address");
    require(amount > 0, "Transfer amount must be greater than zero");

    // Remove fees for transfers to and from charity account or to excluded account
    bool takeFee = true;
    if (FeeAddress == sender || FeeAddress == recipient || _isExcluded[recipient]) {
        takeFee = false;
    }
```

```
        if (!takeFee) removeAllFee();

        if (_isExcluded[sender] && !_isExcluded[recipient]) {
            _transferFromExcluded(sender, recipient, amount);
        } else if (!_isExcluded[sender] && _isExcluded[recipient]) {
            _transferToExcluded(sender, recipient, amount);
        } else if (!_isExcluded[sender] && !_isExcluded[recipient]) {
            _transferStandard(sender, recipient, amount);
        } else if (_isExcluded[sender] && _isExcluded[recipient]) {
            _transferBothExcluded(sender, recipient, amount);
        } else {
            _transferStandard(sender, recipient, amount);
        }

        if (!takeFee) restoreAllFee();
    }

    function _transferStandard(address sender, address recipient, uint256 tAmount) private {
        uint256 currentRate =  _getRate();
        (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount, uint256 tFee,
uint256 tBurn, uint256 tCharity) = _getValues(tAmount);
        uint256 rBurn =  tBurn.mul(currentRate);
        _standardTransferContent(sender, recipient, rAmount, rTransferAmount);
        _sendToCharity(tCharity, sender);
        _reflectFee(rFee, rBurn, tFee, tBurn, tCharity);
        emit Transfer(sender, recipient, tTransferAmount);
    }

    function _standardTransferContent(address sender, address recipient, uint256 rAmount, uint256
rTransferAmount) private {
        _rOwned[sender] = _rOwned[sender].sub(rAmount);
        _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
    }

    function _transferToExcluded(address sender, address recipient, uint256 tAmount) private {
        uint256 currentRate =  _getRate();
        (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount, uint256 tFee,
uint256 tBurn, uint256 tCharity) = _getValues(tAmount);
        uint256 rBurn =  tBurn.mul(currentRate);
        _excludedFromTransferContent(sender, recipient, tTransferAmount, rAmount, rTransferAmount);
        _sendToCharity(tCharity, sender);
        _reflectFee(rFee, rBurn, tFee, tBurn, tCharity);
        emit Transfer(sender, recipient, tTransferAmount);
    }
```

```
    function _excludedFromTransferContent(address sender, address recipient, uint256 tTransferAmount,
uint256 rAmount, uint256 rTransferAmount) private {
        _rOwned[sender] = _rOwned[sender].sub(rAmount);
        _tOwned[recipient] = _tOwned[recipient].add(tTransferAmount);
        _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
    }


    function _transferFromExcluded(address sender, address recipient, uint256 tAmount) private {
        uint256 currentRate =  _getRate();
        (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount, uint256 tFee,
uint256 tBurn, uint256 tCharity) = _getValues(tAmount);
        uint256 rBurn =  tBurn.mul(currentRate);
        _excludedToTransferContent(sender, recipient, tAmount, rAmount, rTransferAmount);
        _sendToCharity(tCharity, sender);
        _reflectFee(rFee, rBurn, tFee, tBurn, tCharity);
        emit Transfer(sender, recipient, tTransferAmount);
    }

    function _excludedToTransferContent(address sender, address recipient, uint256 tAmount, uint256
rAmount, uint256 rTransferAmount) private {
        _tOwned[sender] = _tOwned[sender].sub(tAmount);
        _rOwned[sender] = _rOwned[sender].sub(rAmount);
        _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
    }

    function _transferBothExcluded(address sender, address recipient, uint256 tAmount) private {
        uint256 currentRate =  _getRate();
        (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount, uint256 tFee,
uint256 tBurn, uint256 tCharity) = _getValues(tAmount);
        uint256 rBurn =  tBurn.mul(currentRate);
        _bothTransferContent(sender, recipient, tAmount, rAmount, tTransferAmount, rTransferAmount);
        _sendToCharity(tCharity, sender);
        _reflectFee(rFee, rBurn, tFee, tBurn, tCharity);
        emit Transfer(sender, recipient, tTransferAmount);
    }

    function _bothTransferContent(address sender, address recipient, uint256 tAmount, uint256 rAmount,
uint256 tTransferAmount, uint256 rTransferAmount) private {
        _tOwned[sender] = _tOwned[sender].sub(tAmount);
        _rOwned[sender] = _rOwned[sender].sub(rAmount);
        _tOwned[recipient] = _tOwned[recipient].add(tTransferAmount);
        _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
    }
```

```
    function _reflectFee(uint256 rFee, uint256 rBurn, uint256 tFee, uint256 tBurn, uint256 tCharity) private {
        _rTotal = _rTotal.sub(rFee).sub(rBurn);
        _tFeeTotal = _tFeeTotal.add(tFee);
        _tBurnTotal = _tBurnTotal.add(tBurn);
        _tCharityTotal = _tCharityTotal.add(tCharity);
        _tTotal = _tTotal.sub(tBurn);
                    emit Transfer(address(this), address(0), tBurn);
    }


    function _getValues(uint256 tAmount) private view returns (uint256, uint256, uint256, uint256, uint256,
uint256, uint256) {
        (uint256 tFee, uint256 tBurn, uint256 tCharity) = _getTBasics(tAmount, _TAX_FEE, _BURN_FEE,
_CHARITY_FEE);
        uint256 tTransferAmount = getTTransferAmount(tAmount, tFee, tBurn, tCharity);
        uint256 currentRate =  _getRate();
        (uint256 rAmount, uint256 rFee) = _getRBasics(tAmount, tFee, currentRate);
        uint256 rTransferAmount = _getRTransferAmount(rAmount, rFee, tBurn, tCharity, currentRate);
        return (rAmount, rTransferAmount, rFee, tTransferAmount, tFee, tBurn, tCharity);
    }

    function _getTBasics(uint256 tAmount, uint256 taxFee, uint256 burnFee, uint256 charityFee) private
view returns (uint256, uint256, uint256) {
        uint256 tFee = ((tAmount.mul(taxFee)).div(_GRANULARITY)).div(100);
        uint256 tBurn = ((tAmount.mul(burnFee)).div(_GRANULARITY)).div(100);
        uint256 tCharity = ((tAmount.mul(charityFee)).div(_GRANULARITY)).div(100);
        return (tFee, tBurn, tCharity);
    }

    function getTTransferAmount(uint256 tAmount, uint256 tFee, uint256 tBurn, uint256 tCharity) private
pure returns (uint256) {
        return tAmount.sub(tFee).sub(tBurn).sub(tCharity);
    }

    function _getRBasics(uint256 tAmount, uint256 tFee, uint256 currentRate) private pure returns
(uint256, uint256) {
        uint256 rAmount = tAmount.mul(currentRate);
        uint256 rFee = tFee.mul(currentRate);
        return (rAmount, rFee);
    }

    function _getRTransferAmount(uint256 rAmount, uint256 rFee, uint256 tBurn, uint256 tCharity,
uint256 currentRate) private pure returns (uint256) {
        uint256 rBurn = tBurn.mul(currentRate);
```

```
        uint256 rCharity = tCharity.mul(currentRate);
        uint256 rTransferAmount = rAmount.sub(rFee).sub(rBurn).sub(rCharity);
        return rTransferAmount;
    }

    function _getRate() private view returns(uint256) {
        (uint256 rSupply, uint256 tSupply) = _getCurrentSupply();
        return rSupply.div(tSupply);
    }

    function _getCurrentSupply() private view returns(uint256, uint256) {
        uint256 rSupply = _rTotal;
        uint256 tSupply = _tTotal;
        for (uint256 i = 0; i < _excluded.length; i++) {
            if (_rOwned[_excluded[i]] > rSupply || _tOwned[_excluded[i]] > tSupply) return (_rTotal, _tTotal);
            rSupply = rSupply.sub(_rOwned[_excluded[i]]);
            tSupply = tSupply.sub(_tOwned[_excluded[i]]);
        }
        if (rSupply < _rTotal.div(_tTotal)) return (_rTotal, _tTotal);
        return (rSupply, tSupply);
    }

    function _sendToCharity(uint256 tCharity, address sender) private {
        uint256 currentRate = _getRate();
        uint256 rCharity = tCharity.mul(currentRate);
        _rOwned[FeeAddress] = _rOwned[FeeAddress].add(rCharity);
        _tOwned[FeeAddress] = _tOwned[FeeAddress].add(tCharity);
        emit Transfer(sender, FeeAddress, tCharity);
    }

    function removeAllFee() private {
        if(_TAX_FEE == 0 && _BURN_FEE == 0 && _CHARITY_FEE == 0) return;

        ORIG_TAX_FEE = _TAX_FEE;
        ORIG_BURN_FEE = _BURN_FEE;
        ORIG_CHARITY_FEE = _CHARITY_FEE;

        _TAX_FEE = 0;
        _BURN_FEE = 0;
        _CHARITY_FEE = 0;
    }

    function restoreAllFee() private {
        _TAX_FEE = ORIG_TAX_FEE;
        _BURN_FEE = ORIG_BURN_FEE;
```

```
        _CHARITY_FEE = ORIG_CHARITY_FEE;
    }

    function _getTaxFee() private view returns(uint256) {
        return _TAX_FEE;
    }


}
```

(Dados fornecidos pela CoinTool.app).

ESPECIFICAÇÕES PRÁTICAS DO GAIA TOKEN

GAIA Token suporta processos de DeFi (adicionável a fundos de liquidez em algumas DEX (Descentralized Exchanges), como a MDex, a Pancake Swap, Poocoin e outras.

Sistema atualizado para rodar em ambiente Web3. Possui sistema de recompensas para holders, obtidos através das taxas de compra e venda do token.

Para maior segurança em valores e liquidez, o código fonte prevê variações bruscas de preços de mercado, com uma trava anti baleia (Anti Whale Locker). Na linguagem corrente do mercado, entende-se como “baleia” aquele holder possuidor de muitos tokens; uma venda brusca de todos os seus tokens não afetará muito o percentual de valor da moeda.

Sistema travado de cunhagem de mais moedas. GAIA Token nunca possuirá mais de 25 milhões de unidades, o que garante seu uso como reserva de valor, mantendo-se rara.

Recompensas diretas nas carteiras de holders, através do sistema de proof-of-stake (PoS).

Segurança na transferência de valores, através da blockchain da Binance Smart Chain, de forma a não ser aberta para mineração, garantindo que os lucros das taxas cobradas sejam indexados apenas para possuidores dos tokens, por percentual de propriedade (Quem tem mais tokens, ganha mais recompensas. Como a moeda é muito recente, essas recompensas possuem um alto retorno, mas, tendendo a diminuição, conforme sejam vendidos os tokens até seu limite de suprimento. A partir daí, os ganhos serão diminuídos, mas sempre estarão presentes, já que sempre haverá taxas de gás para compras e vendas, bem como movimentações entre carteiras.

Código fonte travado na fonte. Nem os criadores programadores podem fazer quaisquer alterações, que possam manipular ou por em risco o futuro do GAIA Token.

Além disso, GAIA Token já possui site no ar e comunidades criadas. Para que o possuidor possa se manter informado a todo instante, aconselhamos que os mesmos se inscrevam nessas comunidades e visitem periodicamente nosso site, das conquistas alcançadas e dos futuros projetos, tanto em desenvolvimento, quanto os que venham a ser criados. Essas informações estão detalhadamente abordadas no próximo tópico.

ROADMAP INICIAL DO GAIA TOKEN

Um “Roadmap” é um tipo de esquema de ações que, ou já foram tomadas, ou estão em desenvolvimento no presente momento, bem como ações futuras que desejamos. Serve como apoio aos desenvolvedores e, ao mesmo tempo, constitui informações importantes que serão adicionados ao token, como produtos diversos, sem, no entanto, influenciar ou modificar o código fonte da criação do mesmo.

Por se tratar de um token muito novo, estamos na sequência principal inicial ainda, como desenvolvimento e criação do suprimento do token e suas características de funcionamento na blockchain, criação e exposição do White Paper, abertura a compras e vendas em exchanges descentralizadas, criação do site principal do GAIA Token, abertura das comunidades, abertura e desenvolvimento de modelos de DeFi. Até aqui, estamos com cumprimento de 100% das tarefas.

No atual momento, GAIA Token já foi auditada pela Coin Tool App e o seu símbolo já foi registrado no Imgur e ICO Holder. Os links para comprovação estão abaixo, respectivamente:
https://imgur.com/gallery/QElkIcf
https://icoholder.com/en/gaia-token-1035367

O site oficial do GAIA Token é https://www.gaiatokenecossystem.com

No site, existe um email para contato direto com os desenvolvedores do projeto, que é o contact@gaiatokenecossystem.com . Esse é o email principal. Existe um segundo e terceiro emails, que serviu como suporte para a criação das comunidades nas redes sociais: contact.gaiatokens@gmail.com , e gaia.token.2022@gmail.com .

As comunidades criadas estão listadas abaixo, com seus respectivos links de verificação e cadastro. Mais uma vez, pedimos que os holders e traders de GAIA Token se inscrevam nessas comunidades e visitem o site, a fim de se manterem informados de nossas posições.

Medium: https://medium.com/@gaia.token.2022/gaia-token-gaia-c0bda6d820a0

Github: https://github.com/GaiaToken2022/GAIA-Token-Cryptocurrency

Reddit: https://www.reddit.com/r/Gaia_Tokens/comments/x59kxb/rgaia_tokens_lounge/

Telegram: https://t.me/+XxwKjEucpsc1NWMx

Stack: https://go.stackct.com/app/#/Projects

Facebook: https://m.facebook.com/people/Gaia-Token/100085114159471/?__cft__%5B0%5D=AZUPby_LPggbF0BbTzhMWZAXNvOxj1FL9PV5fv5lI1WOiVsH7CiGl3Dd_oGDBbt2Nk8-ttC9QUKtQLkwpBMZPvN3PyPLJeKIq1sI9FtAupdB3GqT8NltnFEYzIzOs9-_sWOd9Z3Soz01gc2P6vRByPp1yUPDZF02q0_2MKqI1RPJYSP69Mapjp_Xc6y7AS3oNPcPf77q_EsI9EUaCi4S9lgw&__tn__=%3C%3C%2CP-R

LinkedIn: https://www.linkedin.com/feed/?trk=onboarding-landing

Twitter: https://twitter.com/Gaiatokens

Discord: https://discord.com/channels/1032432754379464786/1032432754878591108

BitcoinTalk: https://bitcointalk.org/index.php?topic=2722359.0

Instagram: https://www.instagram.com/gaiatokens/

OpenSea: https://opensea.io/gaiatokenecosystem

ICOholder (Publicação do ícone do Gaia Token): https://icoholder.com/en/gaia-token-1035367

Imgur (Publicação e registro do ícone do GAIA Token) : https://imgur.com/gallery/QElkIcf

Estes são os nossos canais de comunicações oficiais. Também se pode acompanhar a evolução do GAIA Token através dos sites

BSCscan:
https://bscscan.com/token/0x96b978c4f9045f1652cf60f63d73ee3d0e5b737e

Bogged Finance:
https://charts.bogged.finance/?c=bsc&t=0x96b978c4f9045F1652CF60f63d73EE3d0E5b737E

Para obter seus GAIA Token, é preciso configurar sua carteira Metamask, Trustwallet ou outra, na rede Binance Smart Chain, opção Token Personalizado e inserir os seguintes dados de configuração da carteira dedicada aos GAIA Tokens, conforme segue:

Contract address: 0x96b978c4f9045F1652CF60f63d73EE3d0E5b737E
Symbol: GAIA
Decimals: 9

No presente momento, estamos trabalhando para adicionar o ícone do GAIA Token, registrando o mesmo na BSCscan. Em breve, nas buscas pelo token, o ícone deverá aparecer dentro em breve.

FUTUROS PROJETOS DO GAIA TOKEN

Está nos planos dos desenvolvedores do projeto incluir uma série de NFTs dedicados ao GAIA Token. A arrecadação monetária, produto da venda dos mesmos, será revertido a prover mais liquidez para o token, contribuindo para a sua valorização, com o passar do tempo. A divulgação e venda deverá ocorrer em fins de outubro a início de novembro de 2022. Por enquanto, o local de publicação será o OpenSea, com endereço já mencionado no tópico anterior.

Com o crescimento e adesão de mais membros e holders às comunidades e redes sociais de GAIA Token, um token de governança será criado, com dois principais intuitos: agregar valor ao projeto GAIA Token e servir como token de provisão, por parte dos membros, para que a comunidade GAIA tenha poder de voto sobre os destinos futuros de todas as cadeias do projeto.

Com a adesão do token de governança, programado para 2023/2024, o Projeto GAIA Token destinará parte de seus lucros para Organizações Não Governamentais e Instituições que trabalham para recolher animais de rua, cuidando e castrando os mesmos, a fim de que não fiquem abandonados, morram nas ruas ou sofram maus tratos. Este é o motivo da simbologia do ICO do Token ter um gato estilizado. Essa medida será tomada em conjunto com os membros da comunidade, que decidirão, de acordo com as posses de seus tokens de governança, quanto será destinado (deduzido dos lucros obtidos, não ultrapassando o valor deste) e à quais instituições serão contempladas. Essa é mais uma função social do projeto do GAIA Token, uma vez que essa população de cães e gatos de rua são vetores de zoonoses e vírus que, comparando com o Covid-19 e a Varíola dos Macacos, podem migrar para a espécie humana.

Para dar mais estabilidade e solidez ao GAIA Token, um ecossistema de tokens está previsto para ser lançado, gerando maior liquidez, Marketcap e segurança. Todos os tokens lançados terão, cada um, seus White papers e comunidades, além de serem produtos rentáveis e com a mesma solidez e confiabilidade do GAIA Token. Algumas já estão rodando no mercado, mas serão devidamente documentadas no site oficial do ecossistema GAIA: http://gaiatokenecossystem.com.

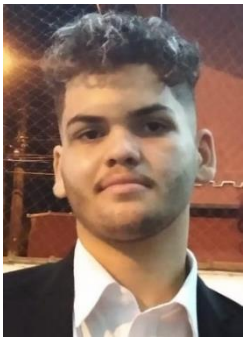
Além desses projetos, previamente decididos, outras ideias e projetos poderão ser agregados ao Roadmap, controlado sempre pela comunidade, tornando-se, assim, um projeto democrático, movido pelo sentimento de melhorar as condições de nosso planeta e da convivência mais harmoniosa da Humanidade, em relação à natureza.

TIME DO PROJETO GAIA TOKEN

Nilton Konnupp
CEO e Programador de softwares

Perfil social LinkedIn:
https://www.linkedin.com/in/nilton-carlos-konnupp-4b0a1627/

Davi Konnupp
Diretor administrativo e Promoter



Perfil social LinkedIn:
https://www.linkedin.com/search/results/all/?heroEntityKey=urn%3Ali%3Afsd_profile%3AACoAAD3ro9gBOHMqPvABaKEm7RcZxFB6HxKYZX4&keywords=davi%20d.&origin=RICH_QUERY_SUGGESTION&position=7&searchId=d3baf215-5c96-4303-9d48-b7ed10751a4e&sid=wEd

Diego Guerra
Diretor de análise de projetos e analista de programação e finanças.

Perfil social LinkedIn:
https://www.linkedin.com/in/diego-guerra-maimone-3a4625239

GAIA

Nossa mascote e inspiradora do projeto.