GPT.sol

```
/**
 *Submitted for verification at Etherscan.io on 2023-03-28
*/

/**
 *Submitted for verification at Arbiscan on 2023-02-28
*/

// SPDX-License-Identifier: Unlicense


pragma solidity 0.8.16;

interface IUniswapV2Factory {
    event PairCreated(
        address indexed token0,
        address indexed token1,
        address pair,
        uint256
    );

    function feeTo() external view returns (address);

    function feeToSetter() external view returns (address);

    function allPairsLength() external view returns (uint256);

    function getPair(address tokenA, address tokenB)
    external
    view
    returns (address pair);

    function allPairs(uint256) external view returns (address pair);

    function createPair(address tokenA, address tokenB)
    external
    returns (address pair);

    function setFeeTo(address) external;

    function setFeeToSetter(address) external;
}

interface IUniswapV2Pair {
    event Approval(
        address indexed owner,
        address indexed spender,
        uint256 value
    );
    event Transfer(address indexed from, address indexed to, uint256 value);

    function name() external pure returns (string memory);

    function symbol() external pure returns (string memory);
```

```
function decimals() external pure returns (uint8);

function totalSupply() external view returns (uint256);

function balanceOf(address owner) external view returns (uint256);

function allowance(address owner, address spender)
external
view
returns (uint256);

function approve(address spender, uint256 value) external returns (bool);

function transfer(address to, uint256 value) external returns (bool);

function transferFrom(
    address from,
    address to,
    uint256 value
) external returns (bool);

function DOMAIN_SEPARATOR() external view returns (bytes32);

function PERMIT_TYPEHASH() external pure returns (bytes32);

function nonces(address owner) external view returns (uint256);

function permit(
    address owner,
    address spender,
    uint256 value,
    uint256 deadline,
    uint8 v,
    bytes32 r,
    bytes32 s
) external;

event Mint(address indexed sender, uint256 amount0, uint256 amount1);
event Burn(
    address indexed sender,
    uint256 amount0,
    uint256 amount1,
    address indexed to
);
event Swap(
    address indexed sender,
    uint256 amount0In,
    uint256 amount1In,
    uint256 amount0Out,
    uint256 amount1Out,
    address indexed to
);
event Sync(uint112 reserve0, uint112 reserve1);

function MINIMUM_LIQUIDITY() external pure returns (uint256);
```

```
    function factory() external view returns (address);

    function token0() external view returns (address);

    function token1() external view returns (address);

    function getReserves()
    external
    view
    returns (
        uint112 reserve0,
        uint112 reserve1,
        uint32 blockTimestampLast
    );

    function price0CumulativeLast() external view returns (uint256);

    function price1CumulativeLast() external view returns (uint256);

    function kLast() external view returns (uint256);

    function mint(address to) external returns (uint256 liquidity);

    function burn(address to)
    external
    returns (uint256 amount0, uint256 amount1);

    function swap(
        uint256 amount0Out,
        uint256 amount1Out,
        address to,
        bytes calldata data
    ) external;

    function skim(address to) external;

    function sync() external;

    function initialize(address, address) external;
}

interface IUniswapV2Router01 {
    function factory() external pure returns (address);

    function WETH() external pure returns (address);

    function addLiquidity(
        address tokenA,
        address tokenB,
        uint256 amountADesired,
        uint256 amountBDesired,
        uint256 amountAMin,
        uint256 amountBMin,
        address to,
        uint256 deadline
    )
    external
```

```
returns (
    uint256 amountA,
    uint256 amountB,
    uint256 liquidity
);

function addLiquidityETH(
    address token,
    uint256 amountTokenDesired,
    uint256 amountTokenMin,
    uint256 amountETHMin,
    address to,
    uint256 deadline
)
external
payable
returns (
    uint256 amountToken,
    uint256 amountETH,
    uint256 liquidity
);

function removeLiquidity(
    address tokenA,
    address tokenB,
    uint256 liquidity,
    uint256 amountAMin,
    uint256 amountBMin,
    address to,
    uint256 deadline
) external returns (uint256 amountA, uint256 amountB);

function removeLiquidityETH(
    address token,
    uint256 liquidity,
    uint256 amountTokenMin,
    uint256 amountETHMin,
    address to,
    uint256 deadline
) external returns (uint256 amountToken, uint256 amountETH);

function removeLiquidityWithPermit(
    address tokenA,
    address tokenB,
    uint256 liquidity,
    uint256 amountAMin,
    uint256 amountBMin,
    address to,
    uint256 deadline,
    bool approveMax,
    uint8 v,
    bytes32 r,
    bytes32 s
) external returns (uint256 amountA, uint256 amountB);

function removeLiquidityETHWithPermit(
    address token,
```

```
    uint256 liquidity,
    uint256 amountTokenMin,
    uint256 amountETHMin,
    address to,
    uint256 deadline,
    bool approveMax,
    uint8 v,
    bytes32 r,
    bytes32 s
) external returns (uint256 amountToken, uint256 amountETH);

function swapExactTokensForTokens(
    uint256 amountIn,
    uint256 amountOutMin,
    address[] calldata path,
    address to,
    uint256 deadline
) external returns (uint256[] memory amounts);

function swapTokensForExactTokens(
    uint256 amountOut,
    uint256 amountInMax,
    address[] calldata path,
    address to,
    uint256 deadline
) external returns (uint256[] memory amounts);

function swapExactETHForTokens(
    uint256 amountOutMin,
    address[] calldata path,
    address to,
    uint256 deadline
) external payable returns (uint256[] memory amounts);

function swapTokensForExactETH(
    uint256 amountOut,
    uint256 amountInMax,
    address[] calldata path,
    address to,
    uint256 deadline
) external returns (uint256[] memory amounts);

function swapExactTokensForETH(
    uint256 amountIn,
    uint256 amountOutMin,
    address[] calldata path,
    address to,
    uint256 deadline
) external returns (uint256[] memory amounts);

function swapETHForExactTokens(
    uint256 amountOut,
    address[] calldata path,
    address to,
    uint256 deadline
) external payable returns (uint256[] memory amounts);
```

```
    function quote(
        uint256 amountA,
        uint256 reserveA,
        uint256 reserveB
    ) external pure returns (uint256 amountB);

    function getAmountOut(
        uint256 amountIn,
        uint256 reserveIn,
        uint256 reserveOut
    ) external pure returns (uint256 amountOut);

    function getAmountIn(
        uint256 amountOut,
        uint256 reserveIn,
        uint256 reserveOut
    ) external pure returns (uint256 amountIn);

    function getAmountsOut(uint256 amountIn, address[] calldata path)
    external
    view
    returns (uint256[] memory amounts);

    function getAmountsIn(uint256 amountOut, address[] calldata path)
    external
    view
    returns (uint256[] memory amounts);
}

interface IUniswapV2Router02 is IUniswapV2Router01 {
    function removeLiquidityETHSupportingFeeOnTransferTokens(
        address token,
        uint256 liquidity,
        uint256 amountTokenMin,
        uint256 amountETHMin,
        address to,
        uint256 deadline
    ) external returns (uint256 amountETH);

    function removeLiquidityETHWithPermitSupportingFeeOnTransferTokens(
        address token,
        uint256 liquidity,
        uint256 amountTokenMin,
        uint256 amountETHMin,
        address to,
        uint256 deadline,
        bool approveMax,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) external returns (uint256 amountETH);

    function swapExactETHForTokensSupportingFeeOnTransferTokens(
        uint256 amountOutMin,
        address[] calldata path,
        address to,
        uint256 deadline
```

```
    ) external payable;

    function swapExactTokensForTokensSupportingFeeOnTransferTokens(
        uint256 amountIn,
        uint256 amountOutMin,
        address[] calldata path,
        address to,
        uint256 deadline
    ) external;

    function swapExactTokensForETHSupportingFeeOnTransferTokens(
        uint256 amountIn,
        uint256 amountOutMin,
        address[] calldata path,
        address to,
        uint256 deadline
    ) external;
}

/**
 * @dev Interface of the ERC20 standard as defined in the EIP.
 */
interface IERC20 {
    /**
     * @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 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 `to`.
     *
     * Returns a boolean value indicating whether the operation succeeded.
```

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

/**
 * @dev Interface for the optional metadata functions from the ERC20
standard.
 *
 * _Available since v4.1._
```

```
 */
interface IERC20Metadata is IERC20 {
    /**
     * @dev Returns the name of the token.
     */
    function name() external view returns (string memory);

    /**
     * @dev Returns the decimals places of the token.
     */
    function decimals() external view returns (uint8);

    /**
     * @dev Returns the symbol of the token.
     */
    function symbol() external view returns (string memory);
}

/**
 * @dev Provides information about the current execution context, including
the
 * sender of the transaction and its data. While these are generally
available
 * via msg.sender and msg.data, they should not be accessed in such a direct
 * manner, since when dealing with meta-transactions the account sending and
 * paying for execution may not be the actual sender (as far as an
application
 * is concerned).
 *
 * This contract is only required for intermediate, library-like contracts.
 */
abstract contract Context {
    function _msgSender() internal view virtual returns (address) {
        return msg.sender;
    }
}

/**
 * @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.
 */
abstract contract Ownable is Context {
    address private _owner;

    event OwnershipTransferred(
```

```
        address indexed previousOwner,
        address indexed newOwner
    );

    /**
     * @dev Initializes the contract setting the deployer as the initial
owner.
     */
    constructor() {
        _transferOwnership(_msgSender());
    }

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        _checkOwner();
        _;
    }

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

    /**
     * @dev Throws if the sender is not the owner.
     */
    function _checkOwner() internal view virtual {
        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 {
        _transferOwnership(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"
```

```
        );
        _transferOwnership(newOwner);
    }

    /**
     * @dev Transfers ownership of the contract to a new account
(`newOwner`).
     * Internal function without access restriction.
     */
    function _transferOwnership(address newOwner) internal virtual {
        address oldOwner = _owner;
        _owner = newOwner;
        emit OwnershipTransferred(oldOwner, newOwner);
    }
}

/**
 * @dev Implementation of the {IERC20} interface.
 *
 * This implementation is agnostic to the way tokens are created. This means
 * that a supply mechanism has to be added in a derived contract using
{_mint}.
 * For a generic mechanism see {ERC20PresetMinterPauser}.
 *
 * TIP: For a detailed writeup see our guide
 * https://forum.openzeppelin.com/t/how-to-implement-erc20-supply-
mechanisms/226[How
 * to implement supply mechanisms].
 *
 * We have followed general OpenZeppelin Contracts guidelines: functions
revert
 * instead returning `false` on failure. This behavior is nonetheless
 * conventional and does not conflict with the expectations of ERC20
 * applications.
 *
 * Additionally, an {Approval} event is emitted on calls to {transferFrom}.
 * This allows applications to reconstruct the allowance for all accounts
just
 * by listening to said events. Other implementations of the EIP may not emit
 * these events, as it isn't required by the specification.
 *
 * Finally, the non-standard {decreaseAllowance} and {increaseAllowance}
 * functions have been added to mitigate the well-known issues around setting
 * allowances. See {IERC20-approve}.
 */
contract ERC20 is Context, IERC20, IERC20Metadata {
    mapping(address => uint256) private _balances;
    mapping(address => mapping(address => uint256)) private _allowances;

    uint256 private _totalSupply;

    string private _name;
    string private _symbol;

    constructor(string memory name_, string memory symbol_) {
        _name = name_;
        _symbol = symbol_;
```

```
    }

    /**
     * @dev Returns the symbol of the token, usually a shorter version of the
     * name.
     */
    function symbol() external view virtual override returns (string memory)
{
        return _symbol;
    }

    /**
     * @dev Returns the name of the token.
     */
    function name() external view virtual override returns (string memory) {
        return _name;
    }

    /**
     * @dev See {IERC20-balanceOf}.
     */
    function balanceOf(address account)
    public
    view
    virtual
    override
    returns (uint256)
    {
        return _balances[account];
    }

    /**
     * @dev Returns the number of decimals used to get its user
representation.
     * For example, if `decimals` equals `2`, a balance of `505` tokens
should
     * be displayed to a user as `5.05` (`505 / 10 ** 2`).
     *
     * Tokens usually opt for a value of 18, imitating the relationship
between
     * Ether and Wei. This is the value {ERC20} uses, unless this function is
     * overridden;
     *
     * NOTE: This information is only used for _display_ purposes: it in
     * no way affects any of the arithmetic of the contract, including
     * {IERC20-balanceOf} and {IERC20-transfer}.
     */
    function decimals() public view virtual override returns (uint8) {
        return 9;
    }

    /**
     * @dev See {IERC20-totalSupply}.
     */
    function totalSupply() external view virtual override returns (uint256) {
        return _totalSupply;
    }
```

```
    /**
     * @dev See {IERC20-allowance}.
     */
    function allowance(address owner, address spender)
    public
    view
    virtual
    override
    returns (uint256)
    {
        return _allowances[owner][spender];
    }

    /**
     * @dev See {IERC20-transfer}.
     *
     * Requirements:
     *
     * - `to` cannot be the zero address.
     * - the caller must have a balance of at least `amount`.
     */
    function transfer(address to, uint256 amount)
    external
    virtual
    override
    returns (bool)
    {
        address owner = _msgSender();
        _transfer(owner, to, amount);
        return true;
    }

    /**
     * @dev See {IERC20-approve}.
     *
     * NOTE: If `amount` is the maximum `uint256`, the allowance is not
updated on
     * `transferFrom`. This is semantically equivalent to an infinite
approval.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    function approve(address spender, uint256 amount)
    external
    virtual
    override
    returns (bool)
    {
        address owner = _msgSender();
        _approve(owner, spender, amount);
        return true;
    }

    /**
```

```
     * @dev See {IERC20-transferFrom}.
     *
     * Emits an {Approval} event indicating the updated allowance. This is
not
     * required by the EIP. See the note at the beginning of {ERC20}.
     *
     * NOTE: Does not update the allowance if the current allowance
     * is the maximum `uint256`.
     *
     * Requirements:
     *
     * - `from` and `to` cannot be the zero address.
     * - `from` must have a balance of at least `amount`.
     * - the caller must have allowance for ``from``'s tokens of at least
     * `amount`.
     */
    function transferFrom(
        address from,
        address to,
        uint256 amount
    ) external virtual override returns (bool) {
        address spender = _msgSender();
        _spendAllowance(from, spender, amount);
        _transfer(from, to, amount);
        return true;
    }

    /**
     * @dev Atomically decreases the allowance granted to `spender` by the
caller.
     *
     * This is an alternative to {approve} that can be used as a mitigation
for
     * problems described in {IERC20-approve}.
     *
     * Emits an {Approval} event indicating the updated allowance.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     * - `spender` must have allowance for the caller of at least
     * `subtractedValue`.
     */
    function decreaseAllowance(address spender, uint256 subtractedValue)
    external
    virtual
    returns (bool)
    {
        address owner = _msgSender();
        uint256 currentAllowance = allowance(owner, spender);
        require(
            currentAllowance >= subtractedValue,
            "ERC20: decreased allowance below zero"
        );
    unchecked {
        _approve(owner, spender, currentAllowance - subtractedValue);
    }
```

```
        return true;
    }

    /**
     * @dev Atomically increases the allowance granted to `spender` by the
caller.
     *
     * This is an alternative to {approve} that can be used as a mitigation
for
     * problems described in {IERC20-approve}.
     *
     * Emits an {Approval} event indicating the updated allowance.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    function increaseAllowance(address spender, uint256 addedValue)
    external
    virtual
    returns (bool)
    {
        address owner = _msgSender();
        _approve(owner, spender, allowance(owner, spender) + addedValue);
        return true;
    }

    /** @dev Creates `amount` tokens and assigns them to `account`,
increasing
     * the total supply.
     *
     * Emits a {Transfer} event with `from` set to the zero address.
     *
     * Requirements:
     *
     * - `account` cannot be the zero address.
     */
    function _mint(address account, uint256 amount) internal virtual {
        require(account != address(0), "ERC20: mint to the zero address");

        _totalSupply += amount;
    unchecked {
        // Overflow not possible: balance + amount is at most totalSupply +
amount, which is checked above.
        _balances[account] += amount;
    }
        emit Transfer(address(0), account, amount);
    }

    /**
     * @dev Destroys `amount` tokens from `account`, reducing the
     * total supply.
     *
     * Emits a {Transfer} event with `to` set to the zero address.
     *
     * Requirements:
```

```
     *
     * - `account` cannot be the zero address.
     * - `account` must have at least `amount` tokens.
     */
    function _burn(address account, uint256 amount) internal virtual {
        require(account != address(0), "ERC20: burn from the zero address");

        uint256 accountBalance = _balances[account];
        require(accountBalance >= amount, "ERC20: burn amount exceeds
balance");
    unchecked {
        _balances[account] = accountBalance - amount;
        // Overflow not possible: amount <= accountBalance <= totalSupply.
        _totalSupply -= amount;
    }

        emit Transfer(account, address(0), amount);
    }

    /**
     * @dev Sets `amount` as the allowance of `spender` over the `owner` s
tokens.
     *
     * This internal function is equivalent to `approve`, and can be used to
     * e.g. set automatic allowances for certain subsystems, etc.
     *
     * Emits an {Approval} event.
     *
     * Requirements:
     *
     * - `owner` cannot be the zero address.
     * - `spender` cannot be the zero address.
     */
    function _approve(
        address owner,
        address spender,
        uint256 amount
    ) internal virtual {
        require(owner != address(0), "ERC20: approve from the zero address");
        require(spender != address(0), "ERC20: approve to the zero address");

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

    /**
     * @dev Updates `owner` s allowance for `spender` based on spent
`amount`.
     *
     * Does not update the allowance amount in case of infinite allowance.
     * Revert if not enough allowance is available.
     *
     * Might emit an {Approval} event.
     */
    function _spendAllowance(
        address owner,
        address spender,
```

```
        uint256 amount
    ) internal virtual {
        uint256 currentAllowance = allowance(owner, spender);
        if (currentAllowance != type(uint256).max) {
            require(
                currentAllowance >= amount,
                "ERC20: insufficient allowance"
            );
        unchecked {
            _approve(owner, spender, currentAllowance - amount);
        }
        }
    }

    function _transfer(
        address from,
        address to,
        uint256 amount
    ) internal virtual {
        require(from != address(0), "ERC20: transfer from the zero address");
        require(to != address(0), "ERC20: transfer to the zero address");

        uint256 fromBalance = _balances[from];
        require(
            fromBalance >= amount,
            "ERC20: transfer amount exceeds balance"
        );
    unchecked {
        _balances[from] = fromBalance - amount;
        // Overflow not possible: the sum of all balances is capped by
totalSupply, and the sum is preserved by
        // decrementing then incrementing.
        _balances[to] += amount;
    }

        emit Transfer(from, to, amount);
    }
}

/**
 * @dev Implementation of the {IERC20} interface.
 *
 * This implementation is agnostic to the way tokens are created. This means
 * that a supply mechanism has to be added in a derived contract using
{_mint}.
 * For a generic mechanism see {ERC20PresetMinterPauser}.
 *
 * TIP: For a detailed writeup see our guide
 * https://forum.openzeppelin.com/t/how-to-implement-erc20-supply-
mechanisms/226[How
 * to implement supply mechanisms].
 *
 * We have followed general OpenZeppelin Contracts guidelines: functions
revert
 * instead returning `false` on failure. This behavior is nonetheless
 * conventional and does not conflict with the expectations of ERC20
 * applications.
```

```
 *
 * Additionally, an {Approval} event is emitted on calls to {transferFrom}.
 * This allows applications to reconstruct the allowance for all accounts
just
 * by listening to said events. Other implementations of the EIP may not emit
 * these events, as it isn't required by the specification.
 *
 * Finally, the non-standard {decreaseAllowance} and {increaseAllowance}
 * functions have been added to mitigate the well-known issues around setting
 * allowances. See {IERC20-approve}.
 */
contract GPT is ERC20, Ownable {
    // TOKENOMICS START
=====================================================>
    string private _name = "GPT";
    string private _symbol = "GPT";
    uint8 private _decimals = 9;
    uint256 private _supply = 5000000000;
    uint256 public taxForMarketing = 4;
    address public marketingFee1 =
0x71880a9cc21B1230459b73cf2E75414315146F5a; //1%
    address public marketingFee2 =
0x0B089a8C9293649950b311eaf53560097a97F8d5; //1%
    address public marketingFee3 =
0x0bFfD4B259F8E6a160B833a0cD25e33C44514809; //2%

    // address public marketingFee1 =
0xAeBE0CBAe9b7Db7de0B4c9796C2FFf46e9D76e7c; //1%
    // address public marketingFee2 =
0xAeBE0CBAe9b7Db7de0B4c9796C2FFf46e9D76e7c; //1%
    // address public marketingFee3 =
0xAeBE0CBAe9b7Db7de0B4c9796C2FFf46e9D76e7c; //2%

    address public DEAD = 0x000000000000000000000000000000000000dEaD;
    uint256 public _marketingReserves = 0;
    mapping(address => bool) public _isExcludedFromFee;
    uint256 public numTokensSellToAddToETH = 100000 * 10 ** _decimals;


    event ExcludedFromFeeUpdated(address _address, bool _status);
    event PairUpdated(address _address);


    // StableDiffusion Access END
================================================>

    IUniswapV2Router02 public immutable uniswapV2Router;
    address public uniswapV2Pair;

    bool inSwapAndLiquify;

    event SwapAndLiquify(
        uint256 tokensSwapped,
        uint256 ethReceived,
        uint256 tokensIntoLiqudity
    );
```

```
    modifier lockTheSwap() {
        inSwapAndLiquify = true;
        _;
        inSwapAndLiquify = false;
    }

    /**
     * @dev Sets the values for {name} and {symbol}.
     *
     * The default value of {decimals} is 18. To select a different value for
     * {decimals} you should overload it.
     *
     * All two of these values are immutable: they can only be set once
during
     * construction.
     */
    constructor() ERC20(_name, _symbol) {
        // _mint(0x5cfBDB8737100561998B15A73b1E1d4A43A263c1, (_supply *
10**_decimals));
        _mint(0x5cfBDB8737100561998B15A73b1E1d4A43A263c1, (_supply * 10 **
_decimals));

        //arbitrum sushi router = 0x1b02dA8Cb0d097eB8D57A175b88c7D8b47997506
        //georli uniswap router = 0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D
        IUniswapV2Router02 _uniswapV2Router =
IUniswapV2Router02(0x1b02dA8Cb0d097eB8D57A175b88c7D8b47997506);
        uniswapV2Pair =
IUniswapV2Factory(_uniswapV2Router.factory()).createPair(address(this),
_uniswapV2Router.WETH());

        uniswapV2Router = _uniswapV2Router;

        _isExcludedFromFee[address(uniswapV2Router)] = true;
        _isExcludedFromFee[msg.sender] = true;
        _isExcludedFromFee[marketingFee1] = true;
        _isExcludedFromFee[marketingFee2] = true;
        _isExcludedFromFee[marketingFee3] = true;
    }

    function updatePair(address _pair) external onlyOwner {
        require(_pair != DEAD, "LP Pair cannot be the Dead wallet, or 0!");
        require(_pair != address(0), "LP Pair cannot be the Dead wallet, or
0!");
        uniswapV2Pair = _pair;
        emit PairUpdated(_pair);
    }

    /**
     * @dev Moves `amount` of tokens from `from` to `to`.
     *
     * This internal function is equivalent to {transfer}, and can be used to
     * e.g. implement automatic token fees, slashing mechanisms, etc.
     *
     * Emits a {Transfer} event.
     *
     * Requirements:
     *
```

```
     *
     * - `from` cannot be the zero address.
     * - `to` cannot be the zero address.
     * - `from` must have a balance of at least `amount`.
     */
    function _transfer(address from, address to, uint256 amount) internal
override {
        require(from != address(0), "ERC20: transfer from the zero address");
        require(to != address(0), "ERC20: transfer to the zero address");
        require(balanceOf(from) >= amount, "ERC20: transfer amount exceeds
balance");

        if ((from == uniswapV2Pair || to == uniswapV2Pair) &&
!inSwapAndLiquify) {
            if (from != uniswapV2Pair) {
                if ((_marketingReserves) >= numTokensSellToAddToETH) {
                    _swapTokensForEth(numTokensSellToAddToETH);
                    _marketingReserves -= numTokensSellToAddToETH;
                    uint256 feeV = address(this).balance;
                    bool sent0 = payable(marketingFee1).send(feeV / 4);
                    //1/4
                    require(sent0, "Failed to send ETH");
                    bool sent1 = payable(marketingFee2).send(feeV / 4);
                    //1/4
                    require(sent1, "Failed to send ETH");
                    bool sent2 = payable(marketingFee3).send(feeV / 2);
                    //2/4
                    require(sent2, "Failed to send ETH");
                }
            }

            uint256 transferAmount;
            if (_isExcludedFromFee[from] || _isExcludedFromFee[to]) {
                transferAmount = amount;
            }
            else {

                uint256 marketingShare = ((amount * taxForMarketing) / 100);
                transferAmount = amount - (marketingShare);
                _marketingReserves += marketingShare;

                super._transfer(from, address(this), marketingShare);
            }
            super._transfer(from, to, transferAmount);
        }
        else {
            super._transfer(from, to, amount);
        }
    }

    function excludeFromFee(address _address, bool _status) external
onlyOwner {
        _isExcludedFromFee[_address] = _status;
        emit ExcludedFromFeeUpdated(_address, _status);
    }
```

```
    function _swapTokensForEth(uint256 tokenAmount) private lockTheSwap {
        address[] memory path = new address[](2);
        path[0] = address(this);
        path[1] = uniswapV2Router.WETH();

        _approve(address(this), address(uniswapV2Router), tokenAmount);

        uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
            tokenAmount,
            0,
            path,
            address(this),
            block.timestamp
        );
    }

    function changeMarketingWallet(address newWallet1, address newWallet2,
address newWallet3)
    public
    onlyOwner
    returns (bool)
    {
        marketingFee1 = newWallet1;
        marketingFee2 = newWallet2;
        marketingFee3 = newWallet3;
        return true;
    }

    function changeTaxForLiquidityAndMarketing(uint256 _taxForMarketing)
    public
    onlyOwner
    returns (bool)
    {
        require((_taxForMarketing) <= 10, "ERC20: total tax must not be
greater than 10%");
        taxForMarketing = _taxForMarketing;

        return true;
    }

    function changeSwapThresholds(uint256 _numTokensSellToAddToETH)
    public
    onlyOwner
    returns (bool)
    {
        require(_numTokensSellToAddToETH < _supply / 98, "Cannot liquidate
more than 2% of the supply at once!");
        numTokensSellToAddToETH = _numTokensSellToAddToETH * 10 ** _decimals;

        return true;
    }

    receive() external payable {}
}
```