

MELIORATE FINANCE

Whitepaper Version 1.0



MELIORATE FINANCE

CONTENTS

Meliorate Finance.....	1
1. INTRODUCTION.....	3
2. CEFI.....	4
3. DEFI	4
4. MTF	5
Meliorate Finance	5
5. MTS	5
Meliorate Swap	5
6. MELIORATE ECOSYSTEM	6
7. MELIORATE WORKFLOW.....	7
8. MELIORATE ON DEFI NETWORK.....	8
9. STAKING.....	9
91 Staking Explained with an example	10
92 Stakingreward Calculation for MTF	10
1) Staking Reward (SR) = $(S - SF) * (DR * D)$	11
2) Un-staking Balance (USR) = $S - [(S * USF) / 100]$	11
10. YIELD FARMING	11
11. LIQUIDITY POOLS.....	14
12. ADDING LIQUIDITY TO EXISTING POOL.....	14
13. ADDING LIQUIDITY TO NEW POOL	14
14. IMPERMANENT LOSS.....	14
15. DEX.....	15
16. MTF WALLET	15
17. PRE-SALE	16
18. MTF TOKEN INFO.....	17
19. ROADMAP.....	18
20. SOCIAL MEDIA.....	19
21. TOKEN ALLOCATIONS.....	20
22. MTF CONTRACT	21

1. INTRODUCTION

Defi's Rise Is Inevitable, and Fusion Is Driving This Evolution of Conventional Finance (Ref - bitcoin.com)

With the inundation of capital in the DeFi space, companies are building more applications for the next generation of financial networks. The DeFi fascination we see in the market right now is therefore helping in our mission to metamorphose the world of money.

🔗 Meliorate is a new blockchain ecosystem which is creating array of DeFi products and a new decentralized financial tool that will allow transparent reward earning/yield farming to its investors. Moreover, Meliorate is a one-stop-shop for all the DeFi needs. 🔗

2. CEFI

Centralized Finance (CeFi) is a service which is structured so that all orders are controlled by one **central** exchange with no other competing parties. The sole aim of CeFi is to make fair trades, boost more transactions, and increase buying and selling processes.

People deposit their money in banks and other financial institutions because they want: 1) to save their money, and 2) to increase those savings by using the fixed and recurring financial instruments available through conventional centralized finance.

One of the problems with this system of centralized finance is that when someone deposits their money in a bank or other institution, they give away control over these assets. They often have little to no knowledge of where their money is actually invested or how it is managed. This lack of transparency turns individuals into nothing more than cogs in an institutionalized financial machine where choice is limited or virtually non-existent.

These centralized institutions use their customers assets to increase their own wealth by investing in different financial markets using a variety of instruments, and by granting loans at high interest rates. We continue to see the large profits regularly made by centralized financial institutions, but only a fraction of those profits are ever returned to depositors.

3. DEFI

DeFi is an abbreviation of the phrase decentralized finance which generally refers to the digital assets and financial smart contracts, protocols, and decentralized applications (DApps) built on Ethereum. In simpler terms, it's financial software built on the blockchain that can be pieced together like Money Legos.

Why do we need it?

Why **do we need DeFi** when **we** have cryptocurrencies? Cryptocurrencies are still dependent on centralized exchanges for their use. **DeFi** brings in decentralized exchanges to make sure that there no centralized point-of-failures within the ecosystem. Centralized organizations manage the majority of the cryptocurrencies.

Privacy, security, and transparency

With DeFi, users obtain control over their wealth, able to securely transact without the need for validation from a central party. As all activities are recorded on the blockchain, all transactions are publicly available and are forever immutable. Thus a transparency and accountability exists in DeFi that eclipses centralized systems, where records can be easily modified.

4. MTF

Meliorate Finance

MTF offers an easy and intuitive way to enter various yield farming opportunities in a few clicks. The token will be constantly adding innovative new pools that, while risky, are pushing composability to its limit to make the most out of what DeFi has to offer.

As a passive participant, depositing tokens to one of MTF's vetted pools, like MTF Staking, MTF/ETH pool will offer a novel way to yield farm with little to no overhead.

Also, the initial burn rate set at 1 - 3% per transaction post token distribution & locked liquidity will make this DeFi a gem.

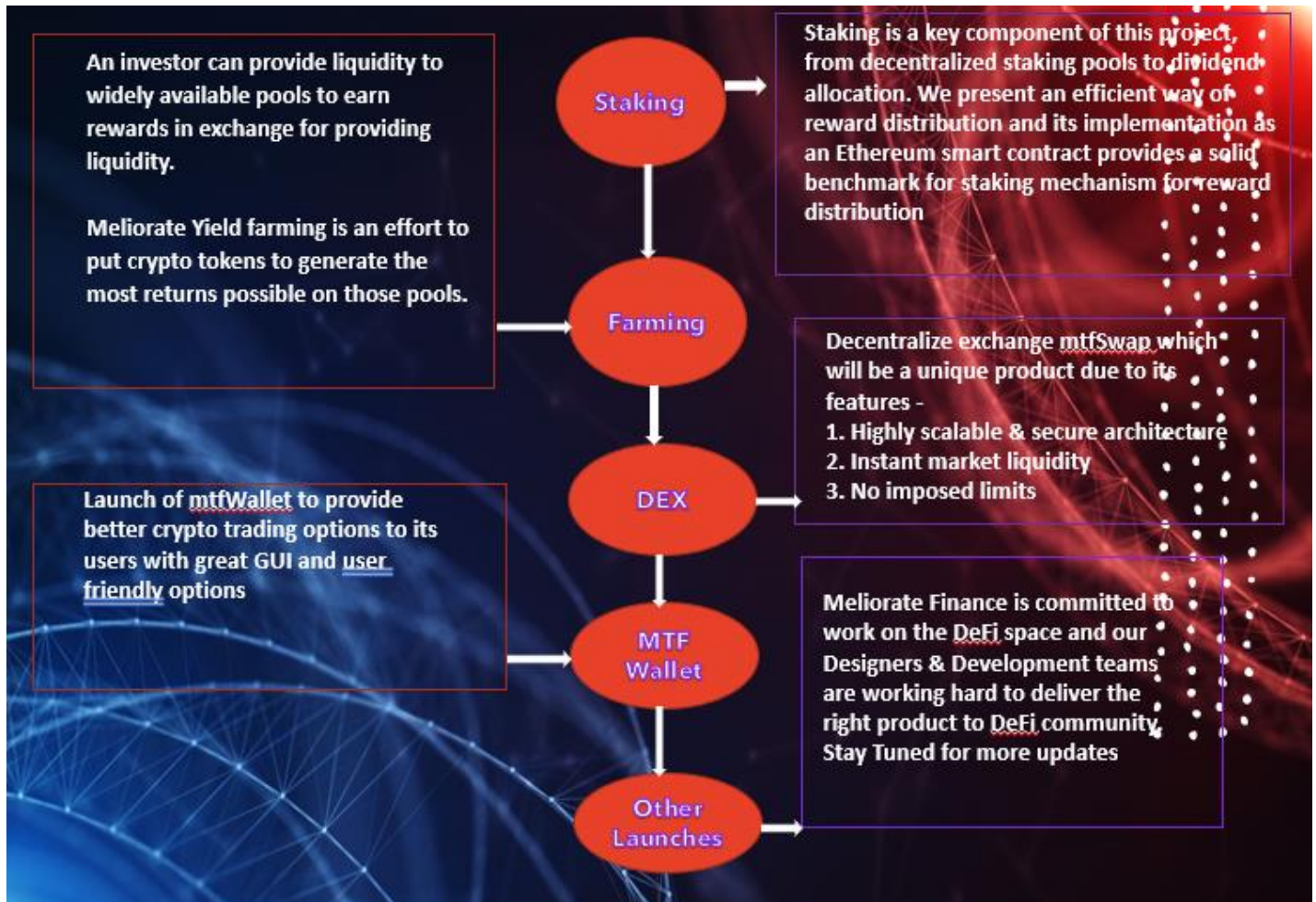
When our contract will go live, most of the raised funds will be sent directly to Uniswap to create a market pair. Our current minimum lockup period is ten months. Our intention is to make sure the ETH/MTF market is always active while the bull run is going.

5. MTS

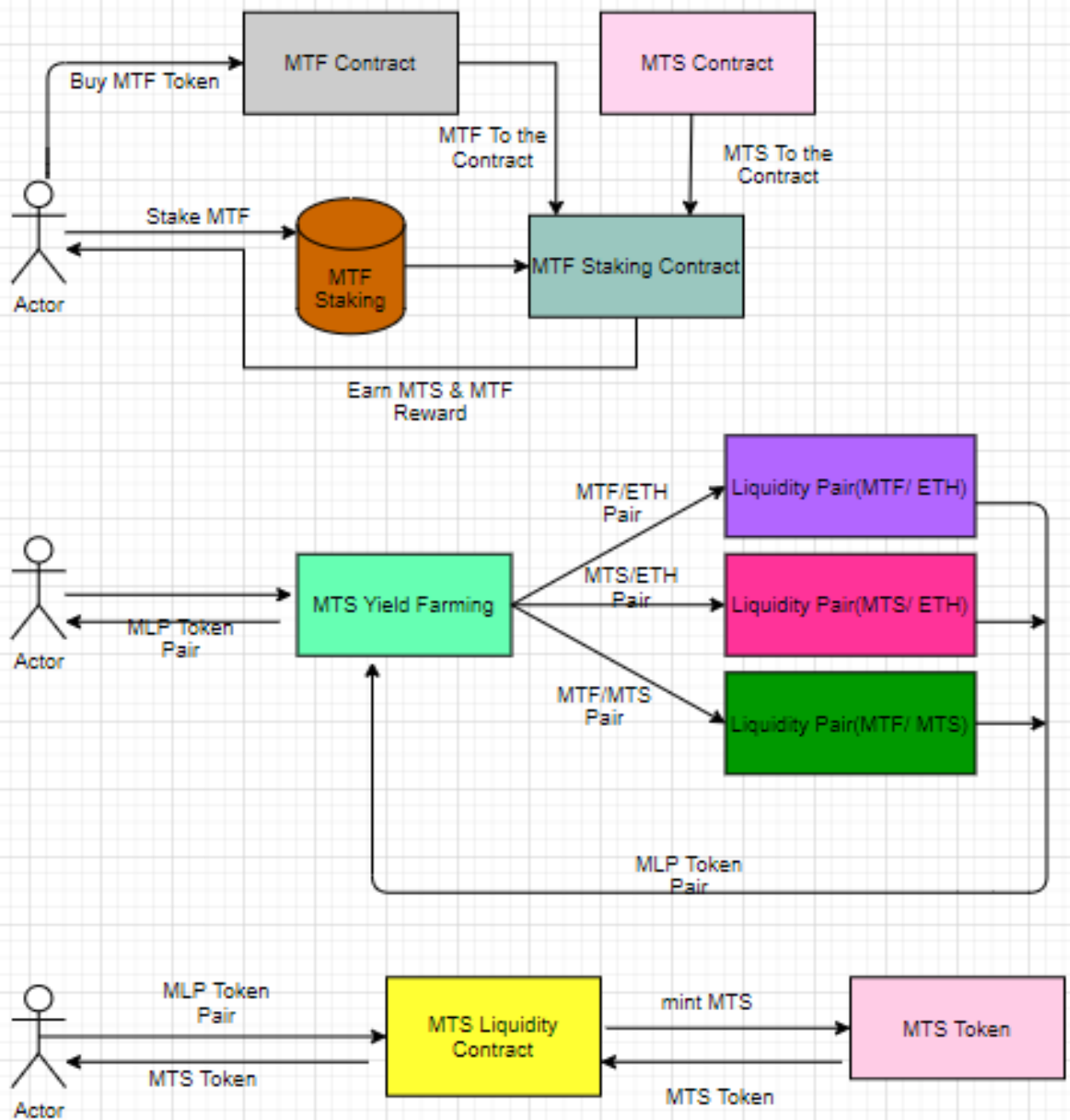
Meliorate Swap

MTS is a swap token from Meliorate Finance. This rare token will be initially available to claim for the staker's of MTF (Meliorate Finance). So, the staker will be able to earn and claim both MTF & MTS. Both the gem will have their own importance in Meliorate ecosystem as Meliorate Finance will be launching array of DeFi products in near future according to roadmap. This gem will be launched for sale once the swap pool is production ready.

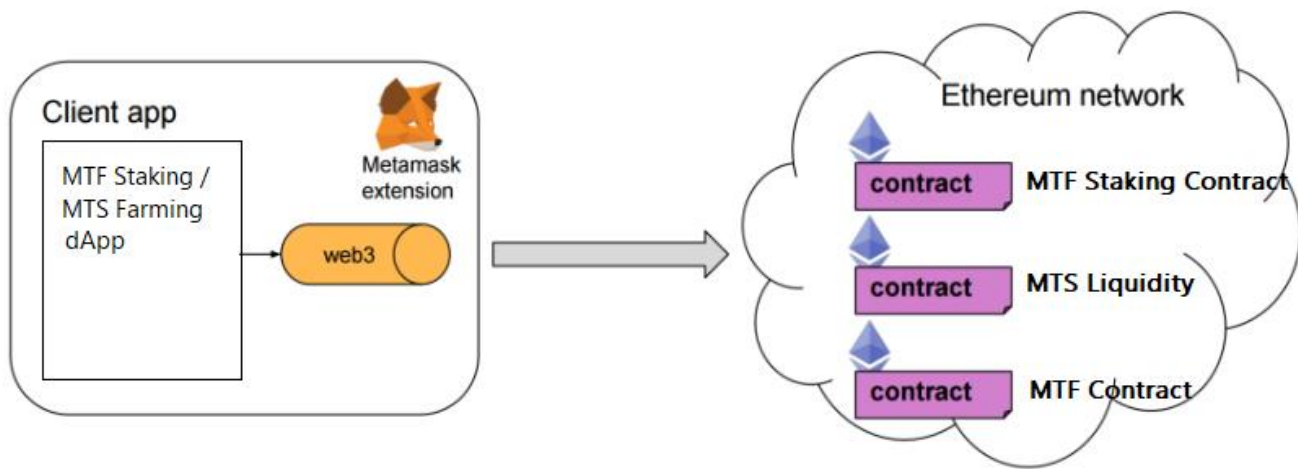
6. MELIORATE ECOSYSTEM



7. MELIORATE WORKFLOW

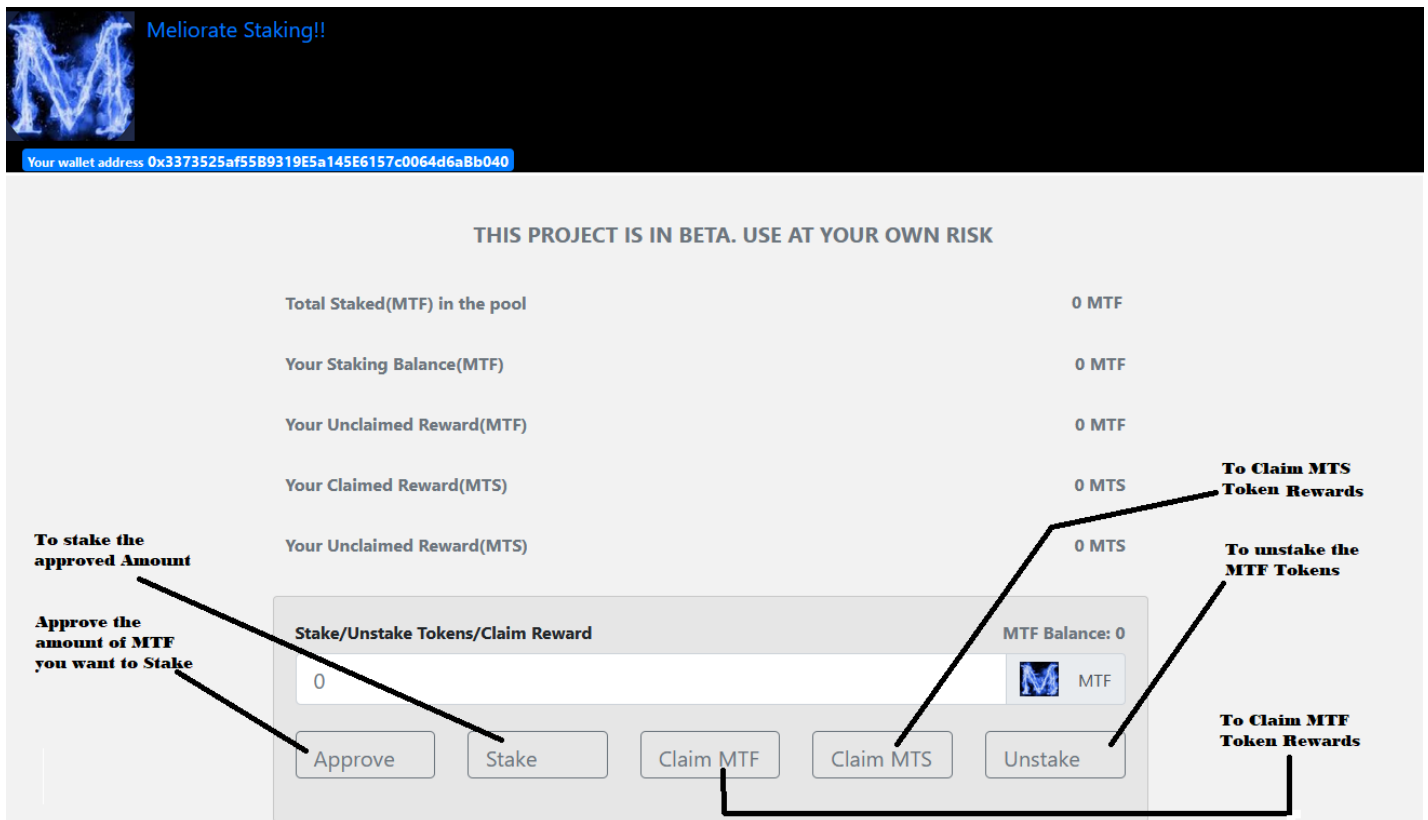


8. MELIORATE ON DEFI NETWORK



9. STAKING

Meliorate Finance will be distributing periodic rewards to its participants in proportion to their stake is a key component of this project, from decentralized staking pools to dividend allocation. We present an efficient way of reward distribution and its implementation as an Ethereum smart contract provides a solid benchmark for staking mechanism for reward distribution. Our implementation scales to any number of registered stake holders and allows for reward distribution events that have higher throughput.



Staking tokens to Meliorate Staking Contract has lot of benefits as mentioned below –

Special offer for Presale Buyers -

The APY (Yearly Percentage Yield) will be ranging between 500-525%. The dividend will be distributed to presale buyers every day until the Presale ends. The dividend rate will change post Uniswap launch.

Example: If Staker stakes 10 MTF tokens

a) Staker will get additional MTF token in the range 0.136 to 0.143 every day(Calculated based on 500-525% APY).

So, if staker stakes 10 MTF for 10 days, staker will earn 1.36 to 1.43 MTF token in 10 days.

b) Staker will get 5 times of MTS token (Meliorate Swap), in this case staker will be able to claim 50 MTS token. \$MTS token is only offered to presale buyers, it will not be available for buyers who buys \$MTF after presale end or the distribution ratio will be reduced.

Staking Benefits post Presale -

- 1) The **APY (Annual Percentage Yield)** will be ranging between 150-200% and dividend distribution will happen every day.
- 2) Staking will deduct 1% of token deposited and distributed to all the stakers.
- 3) Unstaking will deduct 2% of the token deposited and distributed to all the stakers.

91 Staking Explained with an example

- 1) If Staker stakes 100 MTF tokens

1% will be deducted and distributed to all the stakers. So, staked token will be 99

1 token will be distributed to all the stakers

Investor will get 10 times of MTS token, in this case staker will be able to claim 9900 MTS token.

- 2) If Staker Claims or Unstakes MTF token

Investor can claim both MTF and MTS tokens earned as rewards.

Staker can stake MTF token again but this time MTS token will not be available for claim as the MTS reward will be given only once.

- 3) If Staker Unstakes 100 MTF token

2% will be deducted and distributed to all the stakers.

92 Stakingreward Calculation for MTF

Legends:

S= Initial staking balance of MTF

SF: Staking fee = 1%

USF: Unstake fee = 2%

USR = Unstaking calculation

DR = Daily reward = (10 to 15)/30= 0.33 to 0.5 %

D = Staking duration, in Days

Expression:

$$1) \text{ Staking Reward (SR)} = \frac{(S - SF) * (DR * D)}{100}$$

$$2) \text{ Un-staking Balance (USR)} = S - [(S * USF) / 100]$$

10. YIELD FARMING

Meliorate Yield farming will be **more than passive income**. It will provide people the chance to earn investment income by placing funds in a DeFi (decentralized finance) product.

An investor can provide liquidity to widely available pools to earn rewards in exchange for providing liquidity.

Meliorate Yield farming is an effort to put crypto tokens to generate the most returns possible on those pools.

At the simplest level, a yield farmer can move their tokens to different pools to earn more rewards per pool, by identifying whichever pool is offering the best APY on day to day basis.

We incentivise many liquidity pairs by offering our LPs the chance to stake their MLP (Meliorate Liquidity Provider) tokens in our farms. **The Liquidity Pool initially will be comprised of 3 different pools. This will soon increase as per community voting preferences.**

Select Your Favorite Liquidity Pool

Earn MTS tokens by staking UNISWAP V2 MLP Tokens.



MTF-ETH MLP

Deposit MTF-ETH MLP
Earn MTS

Select

APY %




MTS-ETH MLP

Deposit MTS-ETH MLP
Earn MTS

Select

APY %



Meliorate Combo

Deposit MTF-MTS MLP
Earn MTS

Select

APY %

[Telegram](#) [Github](#) [Twitter](#) [Medium](#)



MTF-ETH MLP


Deposit MTF-ETH MLP Tokens and earn MTS



0.000

MTS Earned

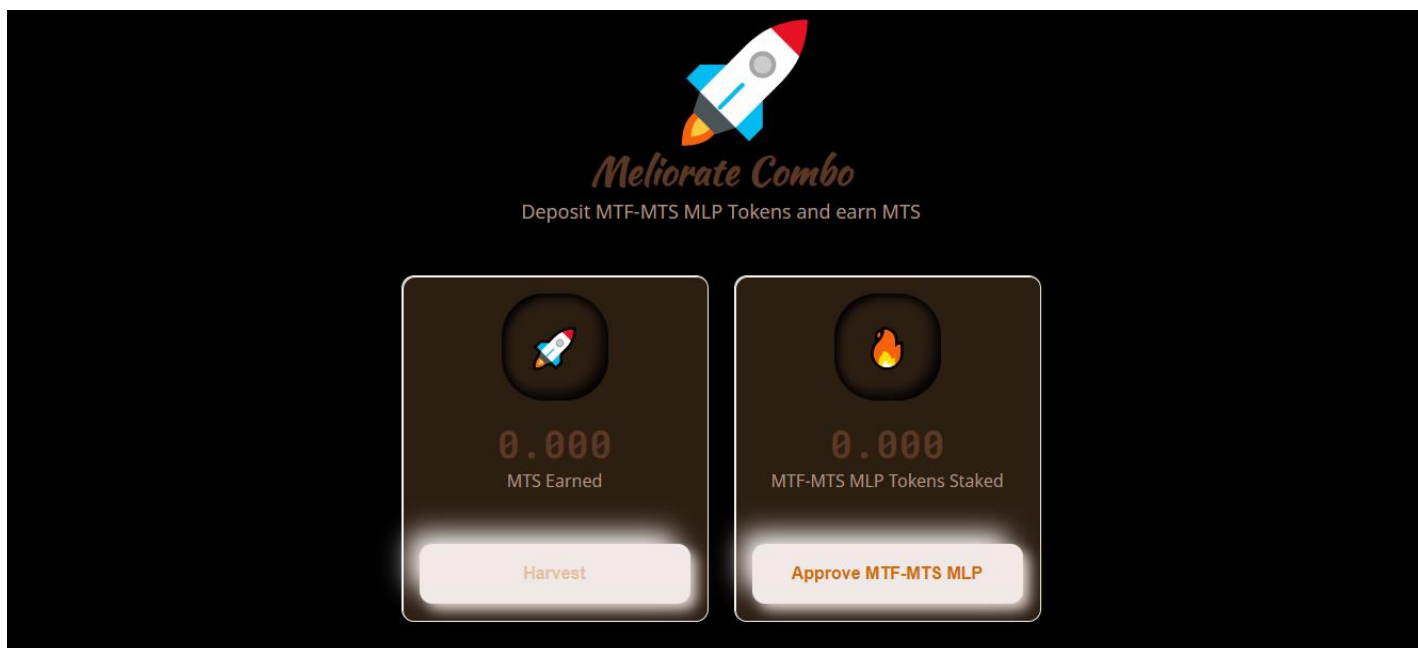
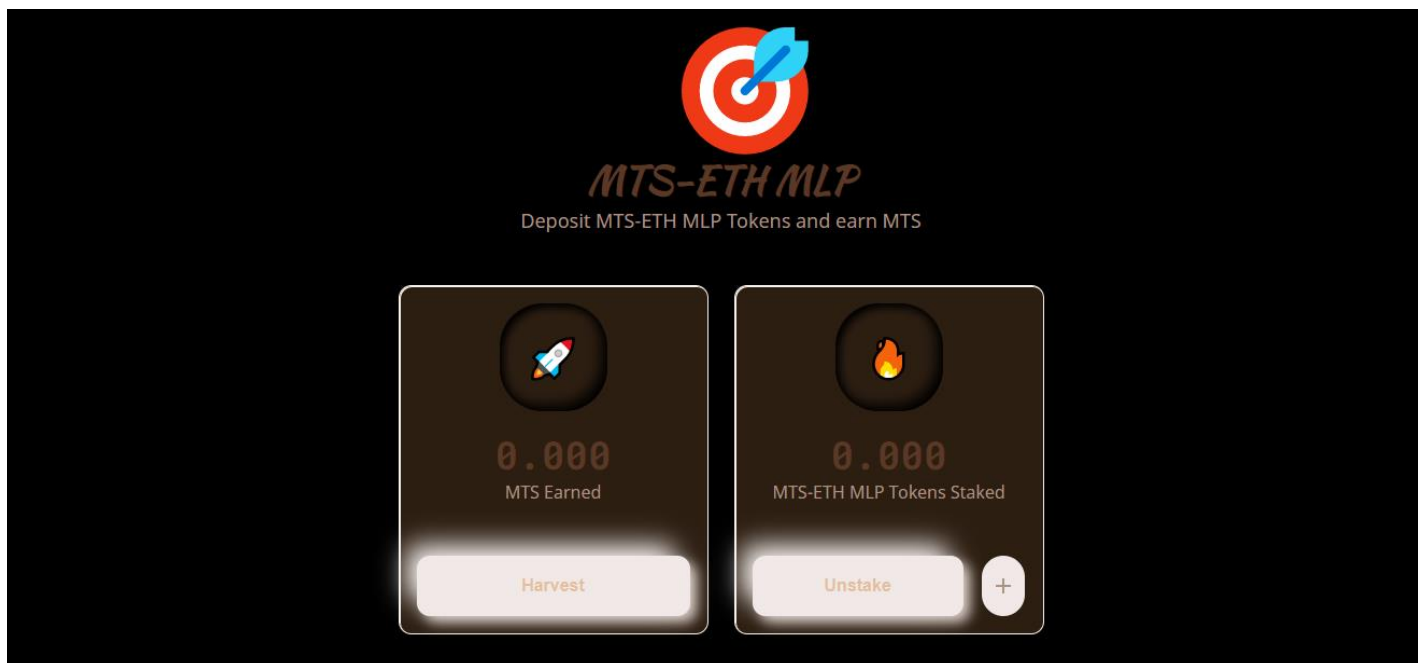
Harvest



0.000

MTF-ETH MLP Tokens Staked

Approve MTF-ETH MLP



APY of each farm will be defined

11. LIQUIDITY POOLS

Meliorate Swap's liquidity pools allow anyone to provide liquidity. When they do so they will receive MLP tokens (Meliorate Swap Liquidity Provider tokens). For example if a user deposited \$MTF and \$ETH into a pool they would receive MTF-ETH MLP tokens. These tokens represent a proportional share of the pooled assets, allowing a user to reclaim their funds at any point.

12. ADDING LIQUIDITY TO EXISTING POOL

You need to provide tokens in a 1:1 ratio to the liquidity pool. This means that if you are adding to, say, a MTF-ETH pool, and wish to provide 2 ETH worth of liquidity, you would need to convert approx 1 ETH to MTF tokens first via our Swap.

13. ADDING LIQUIDITY TO NEW POOL

If the pool you wish to provide liquidity to does not exist, you can create it of course! Just provide the tokens and off you go. As the first liquidity provider, you set the initial exchange ratio (price) if one of the tokens in the pair does not exist yet on UniSwap. This often quickly corrects itself through arbitrage and by more liquidity providers adding to the pool.

14. IMPERMANENT LOSS EXPLAINED

We have found some great article to understand the impermanent loss, please go through with it –

<https://pintail.medium.com/understanding-uniswap-returns-cc593f3499ef>
<https://blog.bancor.network/beginners-guide-to-getting-rekt-by-impermanent-loss-7c9510cb2f22>

They will give you an idea of what IL is and how you are affected by it. Stay tuned for more info!

15. DEX

COMING SOON



16. MTF WALLET

COMING SOON



17. PRE-SALE

1 Presale. 20 MTF/ 1ETH = Presale started on 27th Nov

2 Presale. 17 MTF/ 1ETH = Presale will start on 5th Dec

3 Presale. 15 MTF/ 1ETH = Coming soon/Not planned yet, it'll be decided based on Presale 2nd round token sale.

18. MTF TOKEN INFO

Here is the detailed Tokenomics:

MAX Supply: 20000 MTF

💰 Private Sale: 8000 MTF

Round 1: 3000 MTF => 1ETH/ 20 MTF (Start's on 27th Nov 2020 8:00 AM UTC)

Round 2: 2000 MTF => 1ETH/17 MTF (Start's on 5th Dec 2020 8:00 AM UTC)

Round 3: 3000 MTF => 1ETH/15 MTF (Coming soon)

🔒 Half of the unsold tokens will be kept for staking rewards & Dividend distribution rate will be increased. Rest half of the tokens will be burnt.

✅ Uniswap Liquidity: 3000 MTF

✅ Marketing & Exchange listing: 2500 MTF

✅ Product Development: 500 MTF

🏠 Staking Rewards: 6000 MTF

✅ Uniswap listing: After presale round ends. LIQ will be locked for 18 Months

19. ROADMAP

September 2020(Q3)

Project planning and Team building activity started
 Project Design & roadmap planning
 Contract and Staking dApp development started

November 2020(Q3)

Campaigns for token distribution
 Airdrop
 Private Sale

December 2020(Q3)

Private Sale to continue
 Staking dApp launch → Staking already launched
 Uniswap listing
 Exchange listing

January 2021(Q1)

Yield Farming for Uniswap pairs with multiple options to farm

February - May 2021(Q1-Q2)

Launch of Decentralize exchange mtfSwap

May-Jul(Q2-Q3) 2021

Launch of mtfWallet

What's next ?

Stay tuned for more launches

20. SOCIAL MEDIA

Follow us on our official social media

 Telegram

https://t.me/finance_meliorate

 Twitter

<https://twitter.com/melioratefinan1>

 Github

<https://github.com/MeliorateFinance>

 Medium

<https://Meliorate-finance.medium.com/>

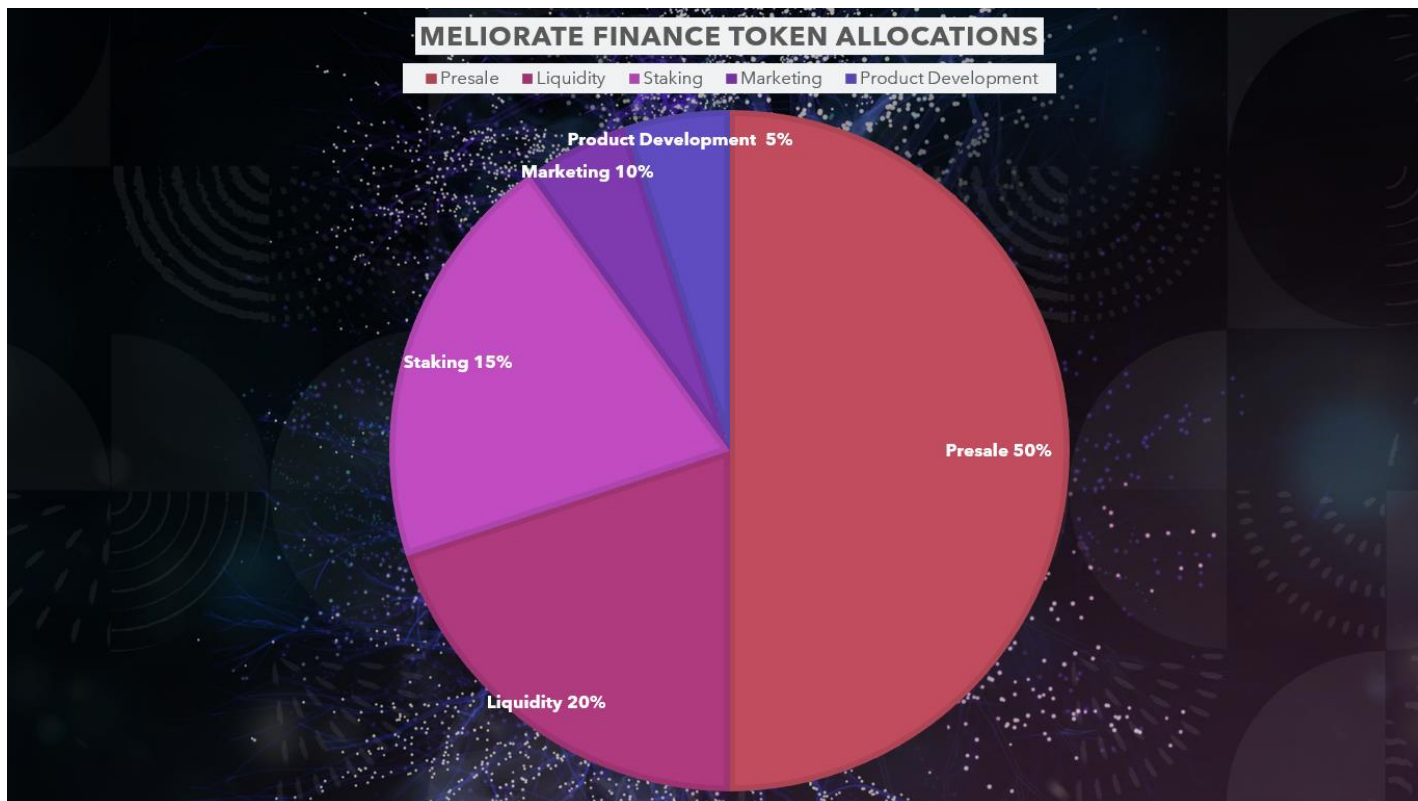
 Website

<https://meliorate.finance>

 Contact

Info@meliorate.finance

21. TOKEN ALLOCATIONS



22. MTF CONTRACT

<https://etherscan.io/address/0xbfd8cc2d30c2f2db71f853d47c6e6ea4fe33d53f>

Contract Address - 0xbfd8cc2d30c2f2db71f853d47c6e6ea4fe33d53f

```
pragma solidity ^0.4.26;
```

```
/**
 * @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 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 c) {
        if (a == 0) {
            return 0;
        }
        c = a * b;
        assert(c / a == b);
        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) {
    // assert(b > 0); // Solidity automatically throws when dividing by 0
    // uint256 c = a / b;
    // assert(a == b * c + a % b); // There is no case in which this doesn't hold
    return a / b;
}

/**
 * @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) {
    assert(b <= a);
    return a - b;
}

/**
 * @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 c) {
    c = a + b;
    assert(c >= a);
    return c;
}
}

contract ClaimableToken {
    /**
     * @dev Returns the amount of tokens owned by `account`.
     */
    function balanceOf(address _owner) constant public 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 _to, uint256 _value) public returns (bool);
}

contract ERC20Basic {

```

```

uint256 public totalSupply;
    /**
    * @dev Returns the amount of tokens owned by `account`.
    */
    function balanceOf(address who) public constant 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 to, uint256 value) public 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);
}

contract ERC20 is ERC20Basic {
    function allowance(address owner, address spender) public constant returns (uint256);
    /**
    * @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 from, address to, uint256 value) public returns (bool);
    /**
    * @dev Sets `amount` as the allowance of `spender` over the `owner`'s tokens.
    *
    * This is 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 spender, uint256 value) public returns (bool);
    /**
    * @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);
}

contract MTFinance is ERC20 {
    using SafeMath for uint256;

```

```

address owner = msg.sender;

mapping (address => uint256) balances;
mapping (address => mapping (address => uint256)) allowed;

string public constant name = "MTFinance";
string public constant symbol = "MTF";
uint public constant decimals = 18;
/**
 * @dev Amount of tokens in existence.
 */
uint256 public totalSupply = 20000e18;
/**
 * @dev Amount of tokens distributed to investors.
 */
uint256 public totalDistributed = 0;
/**
 * @dev Minimum amount of ETH to be invested.
 */
uint256 public constant MIN_CONTRIBUTION = 1 ether / 100; // 0.01 Ether
/**
 * @dev 20 Token per ETH is configured by default which can be amended
 */
uint256 public tokensPerEth = 20e18;
/**
 * @dev The minimum supply can go up to 5000 after burn
 */
uint256 private minimumSupply = 5000e18;
/**
 * @dev Burn rate for transfers, initially set at 0,
 *       which can be changed based on community voting.
 */
uint burnRate=0; // 10 = 1%
/**
 * @dev Event gets fired when the token burn rate changes
 */
event TokenBurnRateUpdated(uint burnRate);
/**
 * @dev Transfet Event gets emmitted when the token transfer gets invoked
 */
event Transfer(address indexed _from, address indexed _to, uint256 _value);
/**
 * @dev Approval Event gets emmitted when the token approval gets invoked
 */
event Approval(address indexed _owner, address indexed _spender, uint256 _value);
/**
 * @dev Distr Event gets emmitted when the token gets distributed to investors, including airdrops
 */
event Distr(address indexed to, uint256 amount);
/**
 * @dev DistrFinished Event gets emmitted when the contract owner ends the presale
 */
event DistrFinished();
/**
 * @dev Airdrop Event gets emmitted when airdrop happens
 */
event Airdrop(address indexed _owner, uint _amount, uint _balance);

```

```

* @dev This event gets emitted when ETH per token is updated
*/
event TokensPerEthUpdated(uint _tokensPerEth);
/**
* @dev This event gets emitted when owner burns the tokens.
*/
event Burn(address indexed burner, uint256 value);
/**
* @dev This flag's value will be changed once presale distribution ends.
*/
bool public distributionFinished = false;

modifier canDistr() {
    require(!distributionFinished);
    _;
}

modifier onlyOwner() {
    require(msg.sender == owner);
    _;
}

/**
* @dev This function is the default constructor which loads on initialization.
*
*/
constructor () public {
    owner = msg.sender;
    distr(owner, totalSupply);
}

/**
* @dev This function can be called to transfer the ownership of the contract.
*
*/
function transferOwnership(address newOwner) onlyOwner public {
    if (newOwner != address(0)) {
        owner = newOwner;
    }
}

/**
* @dev This function can be called in the event of presale End.
*
*/
function finishDistribution() onlyOwner canDistr public returns (bool) {
    distributionFinished = true;
    emit DistrFinished();
    return true;
}

/**
* @dev This function distributes the tokens to the investors.
*
*/
function distr(address _to, uint256 _amount) canDistr private returns (bool) {
    totalDistributed = totalDistributed.add(_amount);
    balances[_to] = balances[_to].add(_amount);
    emit Distr(_to, _amount);
    emit Transfer(address(0), _to, _amount);
}

```

```

    return true;
}
/**
 * @dev This function distributes the tokens to the investors when an airdrop happens.
 *
 */
function startAirdrop(address _participant, uint _amount) internal {

    require( _amount > 0 );

    require( totalDistributed < totalSupply );

    balances[_participant] = balances[_participant].add(_amount);
    totalDistributed = totalDistributed.add(_amount);

    if (totalDistributed >= totalSupply) {
        distributionFinished = true;
    }

    // Emit events for transfer and Airdrop
    emit Airdrop(_participant, _amount, balances[_participant]);
    emit Transfer(address(0), _participant, _amount);
}
/**
 * @dev This function claims remaining MTF tokens to receiver.
 *
 */
function claimMTFTokens(address _participant, uint _amount) public onlyOwner {
    startAirdrop(_participant, _amount);
}
/**
 * @dev This function distributes MTF tokens to multiple receivers.
 *
 */
function distributeAirdropMultipleInvestors(address[] _addresses, uint _amount) public onlyOwner {
    for (uint i = 0; i < _addresses.length; i++)
        startAirdrop(_addresses[i], _amount);
}
/**
 * @dev This function is responsible for updating token rate per ETH
 *
 */
function updateTokensPerEth(uint _tokensPerEth) public onlyOwner {
    tokensPerEth = _tokensPerEth;
    emit TokensPerEthUpdated(_tokensPerEth);
}
/**
 * @dev This is a payable function which gets invoked when ETH is
 *      * sent to this Contract.
 *
 */
function () external payable {
    sendTokens();
}
/**
 * @dev This function gets invoked when ETH is
 *      * sent to this Contract and returns the corresponding MTF token to caller.
 *

```

```

*/
function sendTokens() payable canDistr public {
    uint256 tokens = 0;

    // minimum contribution
    require( msg.value >= MIN_CONTRIBUTION );

    require( msg.value > 0 );

    // get baseline number of tokens
    tokens = tokensPerEth.mul(msg.value) / 1 ether;
    address investor = msg.sender;

    if (tokens > 0) {
        distr(investor, tokens);
    }

    if (totalDistributed >= totalSupply) {
        distributionFinished = true;
    }
}

/**
 * @dev Returns the amount of tokens owned by `account`.
 */
function balanceOf(address _owner) constant public returns (uint256) {
    return balances[_owner];
}

// mitigates the ERC20 short address attack
modifier onlyPayloadSize(uint size) {
    assert(msg.data.length >= size + 4);
    _;
}

/**
 * @dev Emitted when `value` tokens are moved from one account (`from`) to
 * another (`to`).
 *
 * Note that `value` may be zero.
 */
function transfer(address _to, uint256 _amount) onlyPayloadSize(2 * 32) public returns (bool success)
{
    require(_to != address(0));
    require(_amount <= balances[msg.sender]);

    uint256 _tokenToBurn=calculateBurnAmount(_amount);
    uint256 _tokensToTransfer = _amount.sub(_tokenToBurn);

    balances[msg.sender] = balances[msg.sender].sub(_amount);
    balances[_to] = balances[_to].add(_tokensToTransfer);
    totalSupply = totalSupply.sub(_tokenToBurn);
    emit Transfer(msg.sender, _to, _tokensToTransfer);
    emit Transfer(msg.sender, address(0), _tokenToBurn);
    return true;
}

```

```

/**
 * @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 _from, address _to, uint256 _amount) onlyPayloadSize(3 * 32) public
returns (bool success) {

    require(_to != address(0));
    require(_amount <= balances[_from]);
    require(_amount <= allowed[_from][msg.sender]);

    balances[_from] = balances[_from].sub(_amount);

    uint256 _tokenToBurn=calculateBurnAmount(_amount);
    uint256 _tokensToTransfer = _amount.sub(_tokenToBurn);
    balances[_to] = balances[_to].add(_tokensToTransfer);
    totalSupply = totalSupply.sub(_tokenToBurn);
    allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_amount);
    emit Transfer(_from, _to, _tokensToTransfer);
    emit Transfer(_from, address(0), _tokenToBurn);
    return true;
}

/**
 * @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 _value) public returns (bool success) {
    // mitigates the ERC20 spend/approval race condition
    if (_value != 0 && allowed[msg.sender][_spender] != 0) { return false; }
    allowed[msg.sender][_spender] = _value;
    emit Approval(msg.sender, _spender, _value);
    return true;
}

/**
 * @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) constant public returns (uint256) {

```



```

    return allowed[_owner][_spender];
}

/**
 * @dev Returns the token balance of the address WHO for this contract.
 *
 */

function getTokenBalance(address tokenAddress, address who) constant public returns (uint){
    ClaimableToken t = ClaimableToken(tokenAddress);
    uint bal = t.balanceOf(who);
    return bal;
}

/**
 * @dev This function withdraws ETH from the contract.
 *
 */
function withdraw() onlyOwner public {
    address myAddress = this;
    uint256 etherBalance = myAddress.balance;
    owner.transfer(etherBalance);
}

/**
 * @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(uint256 _value) onlyOwner public {
    require(_value <= balances[msg.sender]);
    // no need to require value <= totalSupply, since that would imply the
    // sender's balance is greater than the totalSupply, which *should* be an assertion failure

    address burner = msg.sender;
    balances[burner] = balances[burner].sub(_value);
    totalSupply = totalSupply.sub(_value);
    totalDistributed = totalDistributed.sub(_value);
    emit Burn(burner, _value);
}

/**
 * @dev This function withdraws MTF token from the contract.
 *
 */
function withdrawMTFTokens(address _tokenContract) onlyOwner public returns (bool) {
    ClaimableToken token = ClaimableToken(_tokenContract);
    uint256 amount = token.balanceOf(address(this));
    return token.transfer(owner, amount);
}

/**
 * This function calculates the burn amount to be deducted & burnt on each transfer.

```

```

*
**/
function calculateBurnAmount(uint256 amount) internal view returns (uint256) {
    uint256 _burnAmount = 0;

    // burn amount calculations
    if (totalSupply > minimumSupply) {
        _burnAmount = (amount.mul(burnRate)).div(1000); // Minimum burn is 0
        uint256 availableBurn = totalSupply.sub(minimumSupply);
        if (_burnAmount > availableBurn) { // Only half of the total supply can be burnt
            _burnAmount = availableBurn;
        }
    }

    return _burnAmount;
}

/**
 * @dev Increases/decreases the burn rate
 *
 * Emits a {TokenBurnRateUpdated} event which sets the transfer rate.
 *
 */

    function updateTokenBurnRate(uint _tokensBurnRate) public onlyOwner {
        burnRate = _tokensBurnRate;
        emit TokenBurnRateUpdated(_tokensBurnRate);
    }
}

```

whitepaper version 1.0

meliorate.finance