Whitepaper: GSDI Yield Farming Token (GYFI).

Crypto Shipwright et al., 2021, v. 0.2.5

Citizens, residents, and agents related to businesses, individuals, or other entities considered to be under the jurisdiction of the Securities Exchange Commission of the United States of America are not permitted to hold, transfer, or purchase GYFI tokens. Any accounts believed to be under this jurisdiction will be restricted from receiving GYFI tokens.

Overview

GYFI is the governance token for GSDI and related technologies. Its release transfers the GSDI technology from its developers to the community, increasing the decentralization and security of the protocol. Furthermore the token drives the adoption of GSDI technology by providing rewards to its most dedicated users. GYFI empowers the community to properly market and develop GSDI's future.

GSDIs are a powerful form of debt instruments due to their generic nature and the ability for market mechanisms to set their collateralization and interest rate parameters. For additional details, visit whitepaper.gyfi.eth to view the latest GSDI whitepaper.

As trade in GSDIs requires a two sided market, the greatest obstacle to GSDIs is adoption. The GSDI Yield Farming Token (GYFI) is designed to drive GSDI adoption through lending pools and rewards. GYFI functions as follows:
1) **GSDI Lending Pools:** Users deposit Dai in vaults with strategies which purchase GSDIs;
2) **GYFI Farming:** Strategies selected by governance may be farmed to earn GYFI rewards;
3) **GDAO:** GYFI holders vote in the Snapshot to govern the Gnosis Safe Snap;
4) **Protocol Fee:** A 0.3% GSDI fee may be activated by the GDAO to be distributed to GYFI burners.
5) **Future Benefits:** As 93% of supply is controlled by GDAO and thus GYFI holders, the GDAO may activate many benefits such as more farming rewards, boosts to users who stake or burn GYFI, or other ideas from the community.

Lending Pools

In a GSDI lending pool, users deposit an asset, such as Dai, which is then used by a strategy to purchase GSDI. A strategy is a smart contract which integrates with GSDI dapps to bid on GSDIs meeting certain requirements. For instance, a strategy might integrate with GAUC, the GSDI Auction dapp, to bid on GYFI/ETH GSDIs which have a collateralization ratio below 70%, interest APY above 10%, and a term between 3 and 12 months.

Lending pools are agnostic towards what strategies vaults use and any developer may create their own strategy. Strategies integrate with the IGSDIPool interface. This document includes a sample strategy, GSDI-LP, for GYFI/WETH collateralized bonds purchased from the GAUC auction house dapp.

All GSDI lending pool strategies have two primary risks: duration and undercollateralization. With duration risks, even though the underlying GSDI assets may be worth more than the outstanding Dai deposits in a strategy, the maturity of the GSDIs may be far in the future while Dai withdrawls may be demanded immediately. In undercollateralization, borrowers may default on GSDIs which are worth less than their face value. These two risks collide when depositors in a strategy predict future GSDIs will experience default and withdraw their deposits immediately.

Lending pool strategies are responsible for selecting tradeoffs to manage these risks.
(1) Pools may decide what percentage of Dai assets to maintain for immediate redemptions.

(2) Pools strategies may set their own collateralization expectations for GSDI they purchase.
(3) Pools may require depositors to first request a withdraw of their to give the strategy time to meet redemptions via either sale of GSDI assets or waiting for maturity.

Tokenomics
Total supply: 10m GSDI
92% (9.2m) to GDAO Treasury, to be unlocked over 5 years;
4% (0.4m) as GSDI-LP Vault Farm rewards, to be distributed over 6 months;
2% (0.2m) to be sold in token pool sales for developers;
2% (0.2m) to be sold in token pool sales by GDAO for its treasury.

Token Sales
Token sales will be conducted on a GYFI sale dapp on Ethereum, xDai, and BSC. The sales below are priced in ETH but actual sales will be in ETH, xDai, and BNB for the appropriate chain at the current price ratio. The sales will happen in two parts: (a) five Angel development funding rounds and (b) 2 Series A GDAO treasury rounds.

GYFI token sale idea.
Five 'Angel' rounds for development funding. These funds will be used to compensate the original development team of GSDI and are not in exchange for future work. Development sales total 200k GYFI sold for 250 ETH equivalent of ETH, DAI, & BNB:
1. 40k @ 2000 GYFI/ETH (20 eth) on BSC
2. 40k @ 1000 GYFI/ETH (40 eth) on xDAI
3. 40k @ 800 GYFI/ETH (50 eth) on Ethereum
4. 40k @ 666 GYFI/ETH (60 eth) on BSC
5. 40k @ 500 GYFI/ETH (80 eth) on Ethereum

Each round will last 24 hours or until the target is reached. Participating in a round requires applying for whitelisting. Rounds will begin at a predetermined time.

Four 'Series A' rounds for GDAO treasury. These funds will be used to fill the GDAO treasury and allow it to hire the team needed to develop, market, and operate the project. GDAO sales total 200k GYFI sold for 975 eth equivalent of BNB, xDai, and ETH:
1. 50k @ 400 GYFI/ETH (125 eth) on xDai
2. 50k @ 250 GYFI/ETH (200 eth) on BSC
3. 50k @ 200 GYFI/ETH (250 eth) on Ethereum
4. 50k @ 125 GYFI/ETH (400 eth) on Ethereum

Each round must be scheduled and approved by GDAO.

If a round fails, the tokens will be reassigned to a new round and the sale schedule will be altered. Down rounds will be avoided.

Buyers in the Angel rounds may request GDAO to exit some or all of their position in the Series A Rounds. If these requests are approved, the GDAO may increase the number of tokens available in the Series A Rounds to allow these early buyers to exit since none of the funds raised will be used for DEX or CEX liquidity.

GYFI Interfaces: IGYFIToken

The GYFI token contract inherits from OpenZeppelin's ERC20Snapshot and ERC20Burnable contracts. It is not upgradeable. It includes a blacklist method to prevent accounts from acquiring GYFI which are believed to be under the jurisdiction of the Securities Exchange Commission of the United States of America. This contract is deployed on xDai. The BSC and Ethereum GYFI token contracts are created by xDai's OmniBridge and are their standard ERC20 contracts.

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.8.3 <0.9.0;

/// @title Yield farming token for GSDI.
/// @author Crypto Shipwright
interface IGYFIToken is IERC20 {
    /// @notice Allows the contract owner to blacklist accounts believed to be
under the jurisdiction of the Securities Exchange Commission of the United States
of America. Blacklisted accounts can send but not receive tokens.
    function usaSecJurisdictionBlacklist(address account) external;

    // For ERC677 implementation, see
https://github.com/smartcontractkit/LinkToken/blob/master/contracts/v0.6/
ERC677Token.sol
    // For ERC677 explanation, see https://github.com/ethereum/EIPs/issues/677
    /// @notice Transfers tokens and immediately calls onTokenTransfer on the
receiver.
    function transferAndCall(address to, uint value, bytes memory data) public
virtual returns (bool success);


    // For snapshots, See Open Zeppelin's ERC20 Snapshot.
    /**
     * @dev Emitted by {_snapshot} when a snapshot identified by `id` is created.
     */
    event Snapshot(uint256 id);

    /**
     * @dev Retrieves the balance of `account` at the time `snapshotId` was
created.
     */
    function balanceOfAt(address account, uint256 snapshotId)
        external
        view
        returns (uint256);

    /**
     * @dev Retrieves the total supply at the time `snapshotId` was created.
     */
    function totalSupplyAt(uint256 snapshotId) external view returns (uint256);
}
```

GYFI Contracts: GYFIMasterChef

The GYFIMasterChef is forked from Sushi's MasterChef. The primary differences are GYFIRewardDistributor does not mint tokens and migration is not permitted. Governance is responsible for selecting pools and multipliers for reward distribution.

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.8.3 <0.9.0;
```

```
/// @title Generic Smart Debt Instrument NFTs for lending against generic assets
including vaults. Forked from Sushi's MasterChef.
/// @author Crypto Shipwright
interface IGSDIMasterChef {
    event Deposit(address indexed user, uint256 indexed pid, uint256 amount);
    event Withdraw(address indexed user, uint256 indexed pid, uint256 amount);
    event EmergencyWithdraw(
        address indexed user,
        uint256 indexed pid,
        uint256 amount
    );

    /// @notice Returns the info for a farmer.
    /// @param _pid Unique ID for the pool.
    /// @param _address Address of the farmer.
    /// @return amount_ Amount of tokens.
    /// @return rewardDebt_ Reduces rewards to account for deposits/withdraws. See
Sush's masterchef for details.
    function userInfo(uint256 _pid, uint256 _address)
        external
        view
        returns (uint256 amount_, uint256 rewardDebt_);

    /// @notice Returns the info for a pool.
    /// @param _pid Unique ID for the pool.
    /// @return token_ Token deposited to the pool.
    /// @return allocPoint_ Allocation points assigned to pool, affects percentage
of GYFI to pool.
    /// @return lastRewardBlock_ Last block number that GYFI distribution occured.
    /// @return accGyfiPerShare_ Accumulated GYFIs per share times 10**12.
    function poolInfo(uint256 _pid)
        external
        view
        returns (
            IERC20 token_,
            uint256 allocPoint_,
            uint256 lastRewardBlock_,
            uint256 accGyfiPerShare_
        );

    /// @return bonusEndBlock_ Block at which the current rewards end.
    function bonusEndBlock() external view returns (uint256 bonusEndBlock_);

    /// @return gyfiPerBlock_ GYFI rewards distributed each block.
    function gyfiPerBlock() external view returns (uint256 gyfiPerBlock_);

    /// @return startBlock_ Block at which the rewards begin.
    function startBlock() external view returns (uint256 startBlock_);

    /// @return poolLength_ Total number of pools.
    function poolLength() external view returns (uint256 poolLength_);

    /// @notice Create a new reward pool. Only callable by owner.
    /// @param _allocPoint Allocation points assigned to pool, affects percentage
of GYFI to pool.
    /// @param _token Token deposited to the pool.
    /// @param _withUpdate Whether to update all the pools. Usually should be true
except when saving gas.
```

```
    function add(
        uint256 _allocPoint,
        IERC20 _token,
        bool _withUpdate
    ) external;

    /// @notice Sets the allocation point for the pool. Only callable by owner.
    /// @param _pid Unique ID for the pool.
    /// @param _allocPoint Allocation points assigned to pool, affects percentage
of GYFI to pool.
    /// @param _withUpdate Whether to update all the pools. Usually should be true.
    function set(
        uint256 _pid,
        uint256 _allocPoint,
        bool _withUpdate
    ) external;

    /// @notice The amount of GYFI pending for the farmer in a pool.
    /// @param _pid Unique ID for the pool.
    /// @param _user Address of the farmer.
    /// @return amount_ Amount of GYFI pending for the farmer to claim.
    function pendingGyfi(uint256 _pid, address _user)
        external
        view
        returns (uint256 amount_);

    /// @notice Update the rewards for all pools.
    function massUpdatePools() external;

    /// @notice Update the rewards for one pool.
    /// @param _pid Unique ID for the pool.
    function updatePool(uint256 _pid) external;

    /// @notice Deposit tokens to farm GYFI. Contract must be approved to transfer
tokens from the user.
    /// @param _pid Unique ID for the pool.
    /// @param _amount Amount of tokens to deposit.
    function deposit(uint256 _pid, uint256 _amount) external;

    /// @notice Withdraw tokens from the pool.
    /// @param _pid Unique ID for the pool.
    /// @param _amount Amount of tokens to withdraw.
    function withdraw(uint256 _pid, uint256 _amount) external;

    /// @notice Withdraw deposited tokens without earning any GYFI rewards.
    /// @param _pid Unique ID for the pool.
    function emergencyWithdraw(uint256 _pid) external;
}
```

GYFI Interfaces: IGYFIPool

The IGYFIPool is managed by a strategy which is responsible for its own execution. Users must be careful on which pools they select, as any developer may create a new strategy. Some strategies may even contain centralized components where managers execute transactions. To deploy a new strategy, a IGYFIPool must be deployed alongside the IGYFIStrategy.

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.8.3 <0.9.0;
```

```
/// @title Lending pool for purchasing GSDI via strategies.
/// @author Crypto Shipwright
interface IGYFIPool is IERC20 {
    event Mint(address _staker, uint256 _amountCurrency);
    event Burn(address _staker, uint256 _amountShares);

    /// @notice IGYFIStrategy for the pool.
    /// @return strategy_ Address of the strategy.
    function strategy() external view returns (IGYFIStrategy strategy_);

    /// @notice Mints shares valued at amount. Requires user to approve
IGYFIStrategy for currency.
    /// @dev Uses IGYFIStrategy to find price. Calls IGYFIStrategy.deposit.
    /// @param _amountCurrency Amount of currency in wad to deposit.
    function mint(uint256 _amountCurrency) external;

    /// @notice Burns shares valued at amount. Transfers currency equal to share
value to sender.
    /// @dev Uses IGYFIStrategy to find price. Calls IGYFIStrategy.withdraw.
    function burn(uint256 _amountShares) external;

    /////////////////////////////////////////
    // For snapshots, See the MiniMe token. //
    /////////////////////////////////////////

    /**
     * @dev Retrieves the balance of `account` at the blockNumber.
     */
    function balanceOfAt(address account, uint256 blockNumber)
        external
        view
        returns (uint256);

    /**
     * @dev Retrieves the total supply at the blockNumber.
     */
    function totalSupplyAt(uint256 snapshotId, uint256 blockNumber)
        external
        view
        returns (uint256);
}
```

GYFI Interfaces: IGYFIStrategy

Each pool strategy will have substantial custom code relevant to the particular strategy. This interface only defines the key components required to work with IGYFIPools.

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.8.3 <0.9.0;

/// @title Strategy interface for implementing custom strategies to purchase GSDI.
/// @author Crypto Shipwright
interface IGYFIStrategy {
    enum GSDIStatus {OPEN, COVER, SEIZE, SOLD}

    event GSDIPurchased(uint256 _id);
    event GSDICovered(uint256 _id);
    event GSDISeized(uint256 _id);
    event GSDISold(uint256 _id);
    /// @dev The pool_ must be approved to transfer the currency.
    /// @return pool_ Pool registered with the strategy.
    function pool() external view returns (address pool_);

    /// @return token_ Currenty token that the pool is valued in (such as Dai)
    function currency() external view returns (IERC20 token_);

    /// @dev totalValue should be the sum of expected interest, and total deposits
    /// @dev minus total fees and withdraws. Realized Profits should be added but
may be negative.
    /// @return totalValue_ Total value controlled by the pool.
    function totalValue() external view returns (uint256 totalValue_);

    /// @dev Does not affect total value of the pool. Updated whenever a GSDI is
added or removed.
    /// @return outstandingFaceValue_ Outstanding face value of all GSDI in the
pool.
    function outstandingFaceValue()
        external
        view
        returns (uint256 outstandingFaceValue_);

    /// @dev Adds the outstandingExpectedInterestAt from the most recent to
interestPerSecond divided by the time of that snapshot.
    /// @return outstandingExpectedInterest_ Current expected interest.
    function outstandingExpectedInterest()
        external
        view
        returns (uint256 outstandingExpectedInterest_);

    /// @dev Update realized profit whenever a GSDI is removed. May be negative.
    /// @return realizedProfit_ Current realized profit from covered, seized, and
sold GSDIs.
    function realizedProfit() external view returns (int256 realizedProfit_);

    /// @return totalDeposits_ Total deposits into the strategy in currency.
    function totalDeposits() external view returns (uint256 totalDeposits_);

    /// @return totalWithdraws_ Total withdraws into the strategy in currency.
    function totalWithdraws() external view returns (uint256 totalWithdraws_);

    /// @return totalFees_ Total fees from the strategy in currency.
    function totalFees() external view returns (uint256 totalFees_);

    /// @dev Update whenever a GSDI is added or removed.
    /// @return interestPerSecond_ Current interest per second.
    function interestPerSecond()
        external
        view
        returns (uint256 interestPerSecond_);

    /// @dev Total value divided by outstanding shares multiplied by 10**18.
    /// @return sharePriceWad_ Current price per share in currency times 10**18.
    function sharePriceWad() external view returns (uint256 sharePriceWad_);

    /// @notice Get the info for a GSDI currently held or held in the past by the
strategy.
    /// @dev Caution must be taken if a GSDI is removed and readded later. Most
```

```
strategies should revert.
    /// @param _id ID of the GSDI.
    /// @return purchaseTimestamp_ Timestamp when the GSDI was purchased.
    /// @return purchasePrice_ Price in currency that the GSDI was purchased at.
    /// @return endTimestamp_ Timestamp when the GSDI was removed. Note that this
is when the GSDI's removal was processed, not when it was covered.
    /// @return interestPerSecondWad_ Interest per second times 10**18.
    /// @return profit_ Profit (or loss) in currency when the GSDI was removed.
    /// @return status_ Current status of the GSDI.
    function gsdiInfo(uint256 _id)
        external
        view
        returns (
            uint256 purchaseTimestamp_,
            uint256 purchasePrice_,
            uint256 endTimestamp_,
            uint256 interestPerSecondWad_,
            int256 profit_,
            GSDIStatus status_
        );

    /// @notice Withdraw currency from the contract. Only callable by pool.
    /// @param _amount Amount of currency to withdraw.
    function withdraw(uint256 _amount) external;

    /// @notice Deposit currency to the contract. Only callable by pool.
    /// @param _amount Amount of currency to deposit.
    function deposit(uint256 _amount) external;

    //////////////////////////////////////////
    // For snapshots, See the MiniMe token. //
    //////////////////////////////////////////

    /**
     * @dev Retrieves the interestperSecond at the block number.
     */
    function interestPerSecondAt(address account, uint256 blockNumber)
        external
        view
        returns (uint256 amount_, uint256 timestamp_);

    /**
     * @dev Retrieves the outstandingExpectedInterest at the timestamp.
     */
    function outstandingExpectedInterestAt(address account, uint256 blockNumber)
        external
        view
        returns (uint256 amount_, uint256 timestamp_);
}
```