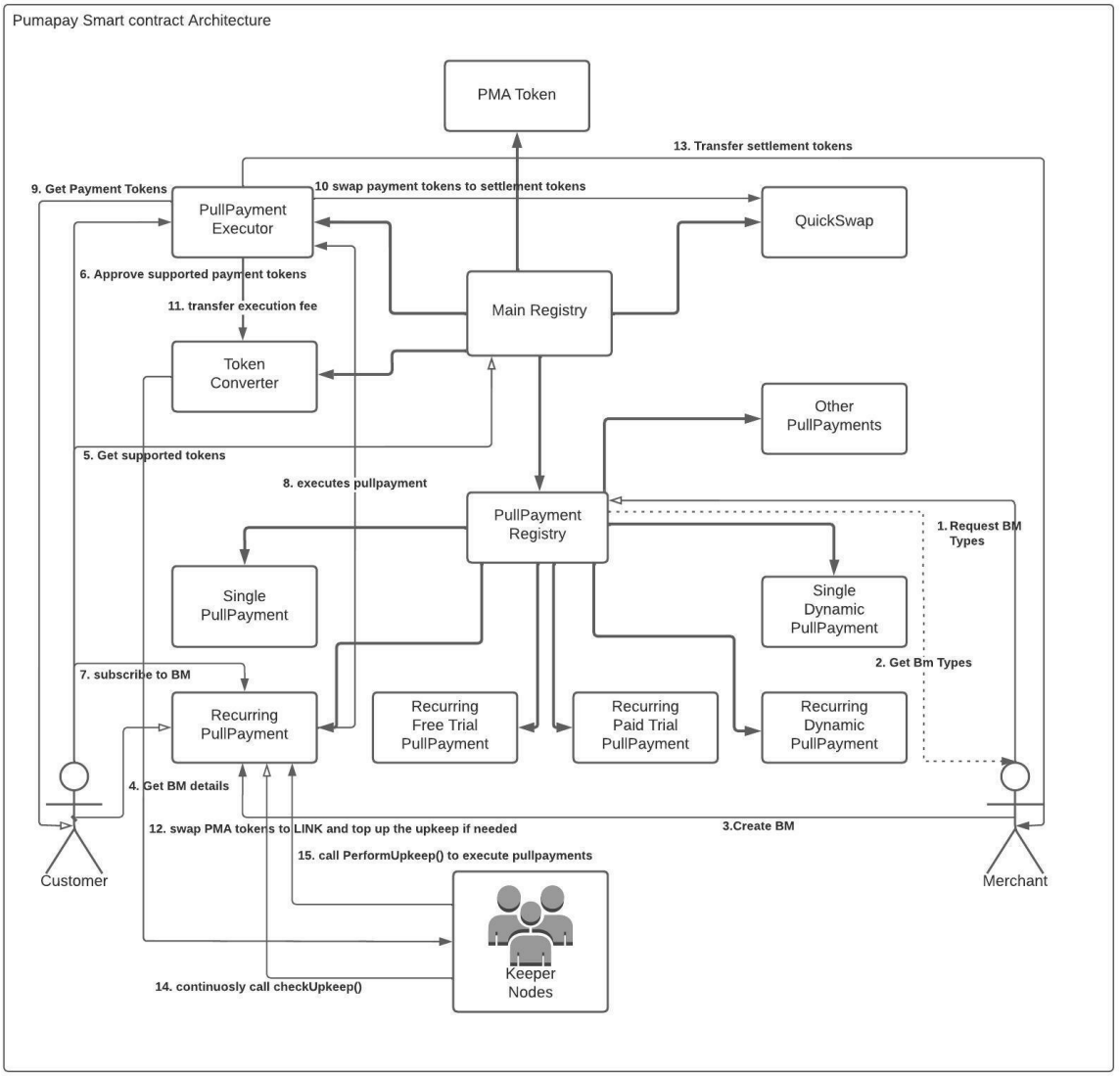
General Smart Contract Architecture

Introduction

The general smart contract architecture describes the components and their interaction in the pumaPay PullPayment system.

Architecture

The following diagram indicates the general smart contract architecture of the pullPayment system.



Components

The general smart contract architecture contains the following components:

1. Main Registry
2. PullPayment Registry
3. PullPayment contracts
4. Executor contract
5. PMA token
6. Swap contract
7. Customers
8. Merchants
9. Keeper Nodes

Main Registry

Introduction

The main Registry contract is the central contract which keeps record of all the contracts in the pullPayment system.

Interface methods

The main registry contains the following interface methods that allows users to interact with the registry:

1. setAddressFor():
2. getAddressForOrDie()
3. getAddressForStringOrDie()
4. getAddressForString()
5. isOneOf()
6. addToken()
7. removeToken()
8. updateExecutionFeeReceiver()
9. updateExecutionFee()
10. updateExtensionPeriod()
11. getSupportedTokens()
12. isSupportedToken()

PullPayment Registry

Introduction

The PullPayment contract is the central contract for the pullPayments which keeps record of all the pullPayment contracts in the system. Users can get the address of any pullPayment contract using this contract. It also stores the low balance subscription ids and the upkeep ids.

Interface methods

The pullPayment Registry contains the following methods that allows users to interact with the registry:

1. grantExecutor()
2. revokeExecutor()
3. addPullPaymentContract()
4. getPPAddressForOrDie()
5. getPPAddressFor()
6. getPPAddressForStringOrDie()
7. getPPAddressForString()
8. isExecutorGranted()
9. addLowBalanceSubscription()
10. removeLowBalanceSubscription()
11. setUpkeepId()

PullPayment Contracts

Introduction

The pullPayments contracts allow the creation and subsctiption of different types of the billing models.

PullPayment contract types:

1. Recurring pullPayment
2. Recurring Free Trial PullPayment
3. Recurring Paid Trial PullPayment
4. Recurring Dynamic PullPayment
5. Single PullPayment
6. Single Dynamic PullPayment

General PullPayment methods:

1. createBillingModel()
2. subscribeToBillingModel()
3. executePullPayment()
4. cancelSubscription()
5. editBillingModel()
6. getBillingModel()
7. getSubscription()
8. getPullPayment()
9. getBillingModel()
10. getBillingModelIdsByAddress()
11. getSubscriptionIdsByAddress()
12. getCanceledSubscriptionIdsByAddress()
13. getPullPaymentIdsByAddress()
14. checkUpkeep()
15. performUpkeep()
16. getSubscriptionIds()
17. isPullPayment()

Executor Contract

Introduction

- The Executor contract allows executors/pullpayment contracts to execute the pullPayments for any subscription Ids.
- It gets the tokens from the customer, swaps the tokens and then transfers the tokens to the merchant.
- It uses the quick swap to swap the tokens.

Interface methods

The executor contracts contain following methods which allows executors to execute the pullPayments.

1. execute()- any user can call this.
2. execute()- Only the user with the executor role can call this.
3. canSwapFromV2()
4. getReceivingAmount()

PMA Token

Introduction

The PMA token is an ERC20 contract. This contract contains the Default admin, minter and pauser roles.

Interface methods

The PMA token contains all the ERC20 methods. Also it contains the-

1. mint()
2. burn()
3. pause()
4. unpause()

Supported Tokens

Introduction

The Supported token contract holds the whitelist of ERC20 tokens that can be used for pullPayments.

Interface Methods

- addToken()
- getSupported()

Swap contract

Introduction

The swap contract uses the pancake swap to swap the tokens. The tokens can be swapped for the following cases:

- Non-PMA -> PMA
- PMA -> Non-PMA
- Non-PMA -> Non-PMA

Customers:

- Customers are the end users of the pullPayment system who subscribes to the particular billing model for the payment.
- Customers interact with the pullPayment registry to request the billing model types and then subscribe to the particular billing model.
- Customers should approve the payment tokens to the Executor contract before subscription.

Merchants:

- The Merchants are the end users of the pullPayment system who creates the billing model for their business.
- Merchants interact with the pullPayment registry to request the billing model types and then create the billing model according to business needs.

Keeper Nodes:

- The chainlink Keeper nodes are responsible for automating the pullpayment execution.
- Keeper nodes continuously call checkUpkeep() method of the upkeep contracts at each block.
- If checkUpkeep() method returns the true boolean value then keeper node calls the performUpkeep() method of the upkeep to execute the batch of pullpaymanets for subscriptions.

Flow of execution

- Before end users start interacting with the system, the system architecture should be in place.
- For that, first we register all the contracts of the system i.e PullPayment Registry, Executor, Swap Contract, PMA token, Supported Tokens etc. in the main Registry.
- Then we register all the pullPayment contracts on the PullPayment Registry.

Once all the smart contracts are in the place, following flow is followed by the pullPayment system-

1. The merchant makes a request to the pullPayment Registry contract for the billing model types.
2. Merchant gets the list of available billing model types.
3. Merchant selects the particular billing model and creates the billing model according to business needs.
4. Merchant receives the billing model id after creating the billing model.
5. Subscriber makes a request to the pullPayment registry contract for the billing model types.
6. Subscriber gets the list of available billing models.
7. Subscriber should approve unlimited tokens to the Executor contract before subscription.
8. Subscriber selects a particular billing model according to needs and subscribes to it.
9. After subscription Subscriber receives the subscription id.
10. The Executors executes the pullPayment for particular subscriptions using the Executor contract.
11. The Executor contract gets the required amount of tokens from the customer, swap the tokens using swap contract and then transfers the tokens to the merchant.