Smart Contract

Security Assessment

For Kanoe

30 Aug 2024

Ascendant

@ascendantfi

www.ascendant.finance

Table of Contents

DISCLAIMER

This independent audit has been conducted to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the codes that were provided for the scope of this audit. This audit report does not constitute agreement, acceptance or advocation for the Project that was audited, and users relying on this audit report should not consider this as having any merit for financial advice in any shape, form or nature. The contracts audited do not account for any economic developments that may be pursued by the Project in question, and that the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are completely free of exploits, bugs, vulnerabilities or deprecation of technologies. Further, this audit report shall not be disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by the auditor.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any persons reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report. Information is provided 'as is', and the auditor is under no covenant to the completeness, accuracy or solidity of the contracts audited. In no event will the auditor or its partners, employees, agents or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this audit report.

Cryptocurrencies and any technologies by extension directly or indirectly related to cryptocurrencies are highly volatile and speculative by nature. All reasonable due diligence and safeguards may yet be insufficient, and users should exercise considerable caution when participating in any shape or form in this nascent industry.

The audit report has made all reasonable attempts to provide clear and articulate recommendations to the Project team with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts provided. It is the sole responsibility of the Project team to sufficiently test and perform checks, ensuring that the contracts are functioning as intended, specifically that the functions therein contained within said contracts have the desired intended effects, functionalities and outcomes of the Project team.

Auditor retains full rights over all intellectual property (including expertise and new attack or exploit vectors) discovered during the audit process. Auditor is therefore allowed and expected to re-use this knowledge in subsequent audits and to inform existing projects that may have similar vulnerabilities. The auditor may, at its discretion, claim bug bounties from third-parties while doing so.

Executive Summary

The smart contracts reviewed in this audit were found to be **Secured**, meaning they contain no critical severity issues that would render them too unsafe to launch. However, it is recommended that the remaining issues found within this report be resolved or mitigated to ensure best user experience.

Security Level	
Well Secured	
Secured	✓
Poorly Secured	
Insecure	

We performed an independent technical audit to identify Smart Contracts uncertainties. This shall protect the code from illegitimate authorization attempts or external & internal threats of any type. This also ensures end-to-end proofing of the contract from frauds. The audit was performed semi-manually. We analyzed the Smart Contracts code line-by-line and used an automation tool to report any suspicious code.

The following tools were used:

- Truffle
- Hardhat
- Remix IDE
- Slither
- Sol2UML

Overview

This report has been prepared for Kanoe for the Ethereum Network. This audit provides a user-centered examination of the smart contracts to look for vulnerabilities, logic errors or other issues from both an internal and external perspective.

Summary

Project Name	Kanoe
Platform	Ethereum
Language	Solidity

Contracts Assessed

Name	Location
Multipay.sol	Not Published
NFT721.sol	Not Published
Plans.sol	Not Published
StableCoin.sol	Not Published
SubscriptionLogic.sol	Not Published
Subscriptions.sol	Not Published
Token.sol	Not Published

Findings Summary

Severity	Found
High	0
Medium	5
Low	5
Informational	44
Total	54

Classification of Issues

High	Exploits, vulnerabilities or errors that will certainly or probabilistically lead towards loss of funds, control, or impairment of the contract and its functions. Issues under this classification are recommended to be fixed with utmost urgency.
Medium	Bugs or issues that may be subject to exploit, though their impact is somewhat limited. Issues under this classification are recommended to be fixed as soon as possible.
Low	Effects are minimal in isolation and do not pose a significant danger to the project or its users. Issues under this classification are recommended to be fixed nonetheless.
Informational	Consistency, syntax or style best practices, Generally pose a negligible level of risk, if any.

Manual Review

Issues Checking Status

Issue Description	Checking Status
Compiler errors	PASS
Race conditions and Reentrancy. Cross-function race conditions.	PASS
Possible delays in data delivery.	PASS
Oracle calls.	PASS
Front running.	PASS
Timestamp dependence.	PASS
Integer Overflow and Underflow.	PASS
DoS with Revert.	PASS
DoS with block gas limit.	PASS
Methods execution permissions.	PASS
Economy model of the contract.	PASS
The impact of the exchange rate on the logic.	PASS
Private user data leaks.	PASS
Malicious Event log.	PASS
Scoping and Declarations.	PASS
Uninitialized storage pointers.	PASS

Arithmetic accuracy.	PASS
Design Logic.	PASS
Cross-function race conditions.	PASS
Safe Open Zeppelin contracts implementation and usage.	PASS
Fallback function security.	PASS

Severity	**Medium**
Contract	Multipay.sol
Description	Denial of Service Vulnerability reserveGas: ln 75

Code Snippet	...
	
	(bool success,) = paymentHolder.call{value: msg.value}("");
	if (!success) {
	revert("Payment failed");
	}
	
	paymentIds.increment();
	payments.push(
	GasPayment(
	paymentIds.current(),
	requestId,
	msg.value,
	block.timestamp,
	msg.sender
	)
	);

Recommendation	A DoS can occur when the caller refuses to accept transfer. Here, the transfer is made before the state is upgraded, potentially allowing the attacker to call the transfer several times before updating the balance. PaymentHolder is presumably an owner-controlled wallet, but if it were somehow replaced by a smart contract, this contract could exploit this function.
Status	

Severity	**Medium**
Contract	Multipay.sol
Description	Checks-Effects-Interactions: sendInheritance: ln 164

Code Snippet	...
	
	sendAmountTo(
	account,
	summaForAll,
	percents,
	successors,
	token
	);
	if (tokensLimit > 0) {
	tokensLimit -= summaForAll;
	if (tokensLimit == 0) {
	break;
	}
	}

Recommendation	Function does not follow checks-effects-interactions flow, which requires state updates to come before transfers. This makes the function vulnerable to reentrancy attack, but can be easily mitigated with a Reentrancy Guard.
Status	

Severity	**Medium**
Contract	Multipay.sol
Description	Checks-Effects-Interactions: sendAmountTo: ln 241

Code Snippet	...
	
	token.safeTransferFrom(account, successors[j], amountToSend);
	
	remains -= amountToSend;

Recommendation	Function does not follow checks-effects-interactions flow, which requires state updates to come before transfers. This makes the function vulnerable to reentrancy attack, but can be easily mitigated with a Reentrancy Guard.
Status	

Severity	**Low**
Contract	Multipay.sol
Description	No Zero Check toggleAdmin: ln 303

Code Snippet	function toggleAdmin(address newAdmin, bool isEnabled) external onlyOwner {
	admins[newAdmin] = isEnabled;
	emit AdminToggled(newAdmin, isEnabled);
	}

Recommendation	The function does not check if the newAdmin is the zero address, which could potentially render the contract inoperable
Status	

Severity	**Low**
Contract	Subscriptions.sol
Description	No Zero Check changeAllowed: ln 204

Code Snippet	function changeAllowed(
	address _manager,
	bool isAllowed
	) external onlyAllowed {
	allowedManager[_manager] = isAllowed;
	}

Recommendation	The function does not check if the new allowed manager is the zero address, which could potentially render the contract inoperable
Status	

Severity	Medium
Contract	SubscriptionLogic.sol
Description	Checks-Effects-Interactions: payExtend: ln 117
Code Snippet	... IERC20(plan.paytokenAddress).transferFrom(account, paymentHolder, plan.paytokenPrice); subscriptionsContract.addPayment(account, 1, subscription.planId, plan.paytokenPrice, plan.paytokenAddress);
Recommendation	Function does not follow checks-effects-interactions flow, which requires state updates to come before transfers. This makes the function vulnerable to reentrancy attack, but can be easily mitigated with a Reentrancy Guard.
Status	

Severity	**Medium**
Contract	SubscriptionLogic.sol
Description	Checks-Effects-Interactions: pay: ln 145
Code Snippet	... IERC20(plan.paytokenAddress).transferFrom(msg.sender, paymentHolder, amount); subscriptionsContract.addPayment(msg.sender, periodsCount, planId, amount, plan.paytokenAddress);
Recommendation	Function does not follow checks-effects-interactions flow, which requires state updates to come before transfers. This makes the function vulnerable to reentrancy attack, but can be easily mitigated with a Reentrancy Guard.
Status	

Severity	**Low**
Contract	SubscriptionLogic.sol
Description	Ignores Return Value payExtend: ln 117

Code Snippet	function payExtend(address account) external onlyAdmin {
	  SubscriptionStruct memory subscription = subscriptionsContract
	  .getUserSubscription(account);
	  PlanStruct memory plan = plansContract.getPlanById(subscription.planId);
	
	IERC20(plan.paytokenAddress).transferFrom(
	  account,
	  paymentHolder,
	  plan.paytokenPrice
	  );
	...

Recommendation	The return value of an external transferFrom call is not checked. Ensure that function returns false in case of a revert.
Status	

Severity	Low
Contract	SubscriptionLogic.sol
Description	Ignores Return Value pay: ln 145

Code Snippet	function pay(uint256 planId, uint256 periodsCount) external {
	  PlanStruct memory plan = plansContract.getPlanById(planId);
	  uint256 amount = plan.paytokenPrice * periodsCount;
	
	IERC20(plan.paytokenAddress).transferFrom(
	  msg.sender,
	  paymentHolder,
	  amount
	  );

Recommendation	The return value of an external transferFrom call is not checked. Ensure that function returns false in case of a revert.
Status	

Severity	**Low**
Contract	SubscriptionLogic.sol
Description	No Zero Check changeAllowed: ln 203

Code Snippet	function changeSystemWallet(address newSystemWallet) external onlyOwner
	{
	address oldSystemWallet = systemWallet;
	systemWallet = newSystemWallet;
	emit SystemWalletChanged(oldSystemWallet, newSystemWallet);
	}

Recommendation	The function does not check if the new system wallet is the zero address, which could potentially render the contract inoperable
Status	

Functional Test Status

Function Name	Type/Return Type	Score
MultiPay		
-From Ownable		
_checkOwner	internal	PASS
_transferOwnership	internal	PASS
owner	public	PASS
renounceOwnership	public	PASS
transferOwnership	public	PASS
-From Context		
_contextSuffixLength	internal	PASS
_msgData	internal	PASS
_msgSender	public	PASS
-From MultiPay		
changePaymentHolder	external	PASS
checkInfo	external	PASS
checkNftCollectionShortInfo	external	PASS
getMaxPaymentId	public	PASS
getUpdates	external	PASS
reserveGas	external	PASS
sendAmountTo	internal	PASS

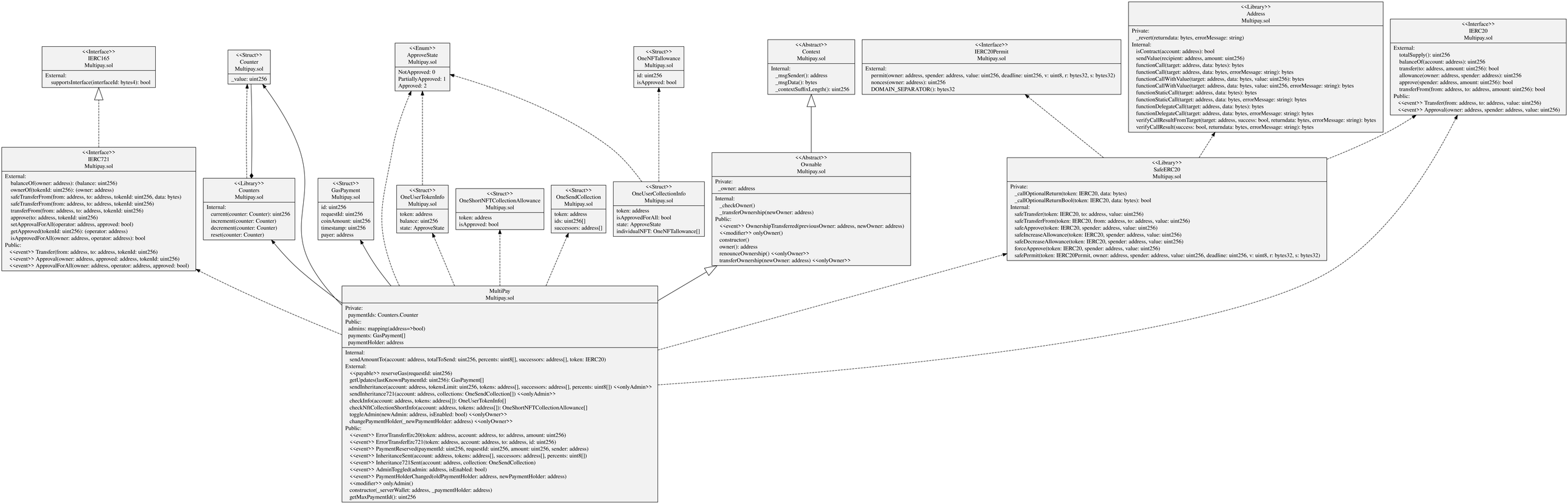
sendInheritance	external	PASS
sendInheritance721	external	PASS
toggleAdmin	external	PASS
Plans		
addPlan	external	PASS
getPlanById	external	PASS
getPlanId	external	PASS
getPlansList	external	PASS
togglePlan	external	PASS
Subscriptions		
-From Counters		
current	internal	PASS
decrement	internal	PASS
increment	internal	PASS
reset	internal	PASS
-From Subscriptions		
addPayment	external	PASS
addSubscription	private	PASS
changeAllowed	external	PASS
extendSubscriptionById	private	PASS
getMaxPaymentId	external	PASS
getMaxSubscriptionId	external	PASS
getPaymentsList	external	PASS

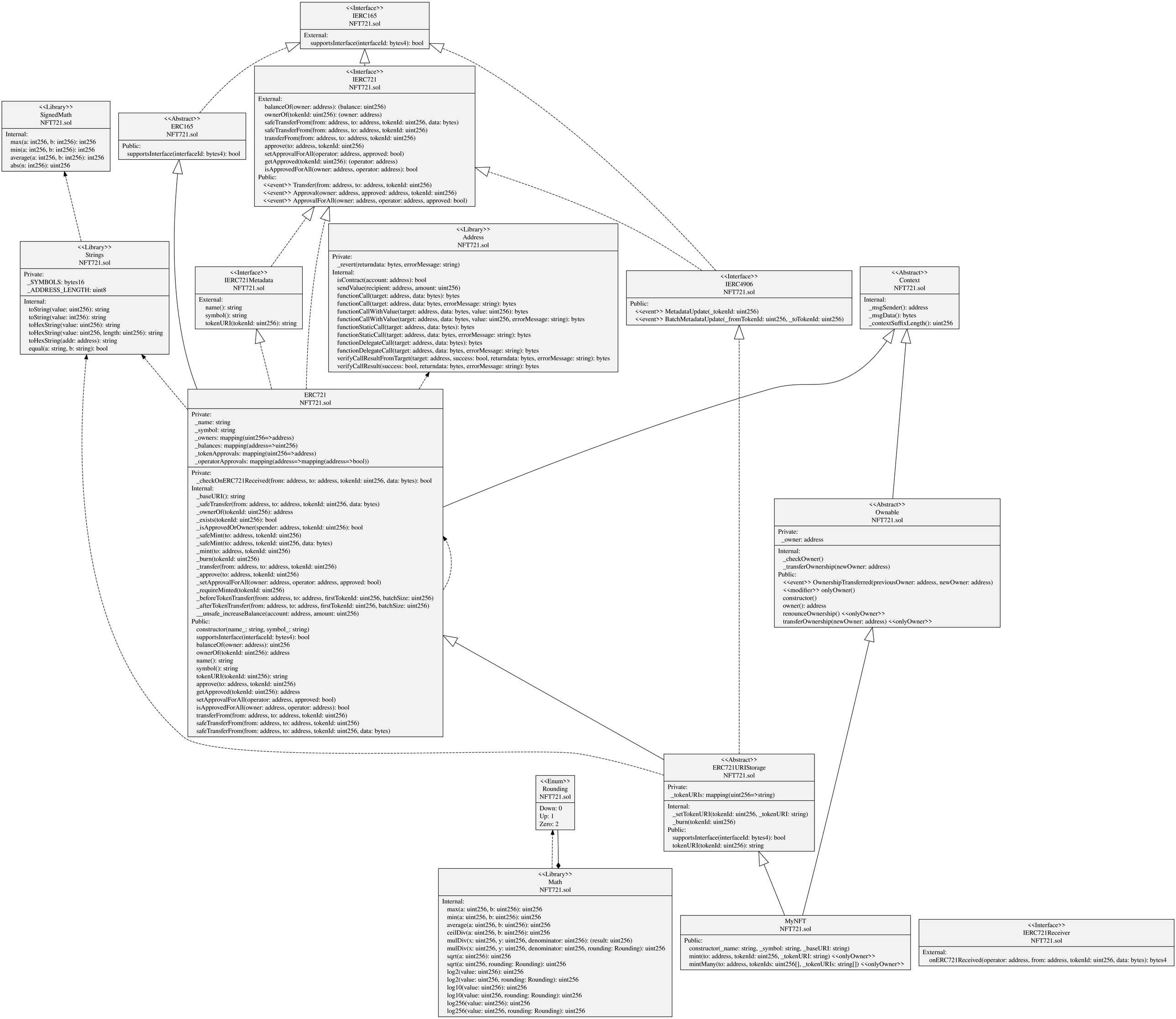
getSubscriptionById	public	PASS
getSubscriptionsList	external	PASS
getUserSubscription	external	PASS
SubscriptionLogic		
changePlansContract	external	PASS
changeSubscriptionContract	external	PASS
getUpdates	external	PASS
getUserSubscription	external	PASS
pay	external	PASS
payExtend	external	PASS
NFT721		
-From IERC165		
mint	public	PASS
mintMany	public	PASS
StableCoin		
decimals	public	PASS
constructor	public	PASS
Token		
constructor	public	PASS

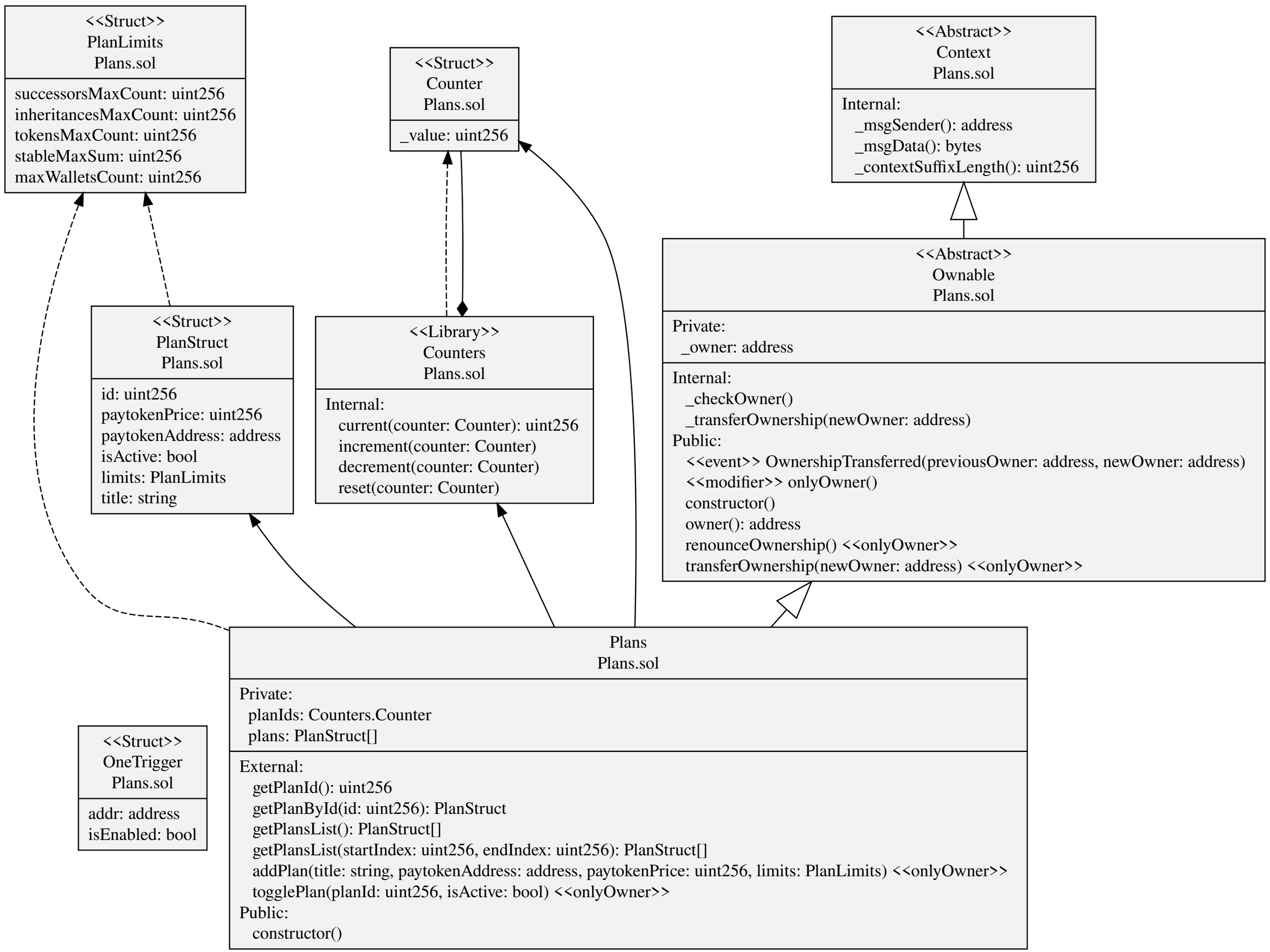
Omitted Results

Note: Any issues that have been omitted from this report have been deemed by the reviewing team as irrelevant, inapplicable, and/or negligible to the proper functioning of this contract. Thus, any omitted issues can be safely ignored.

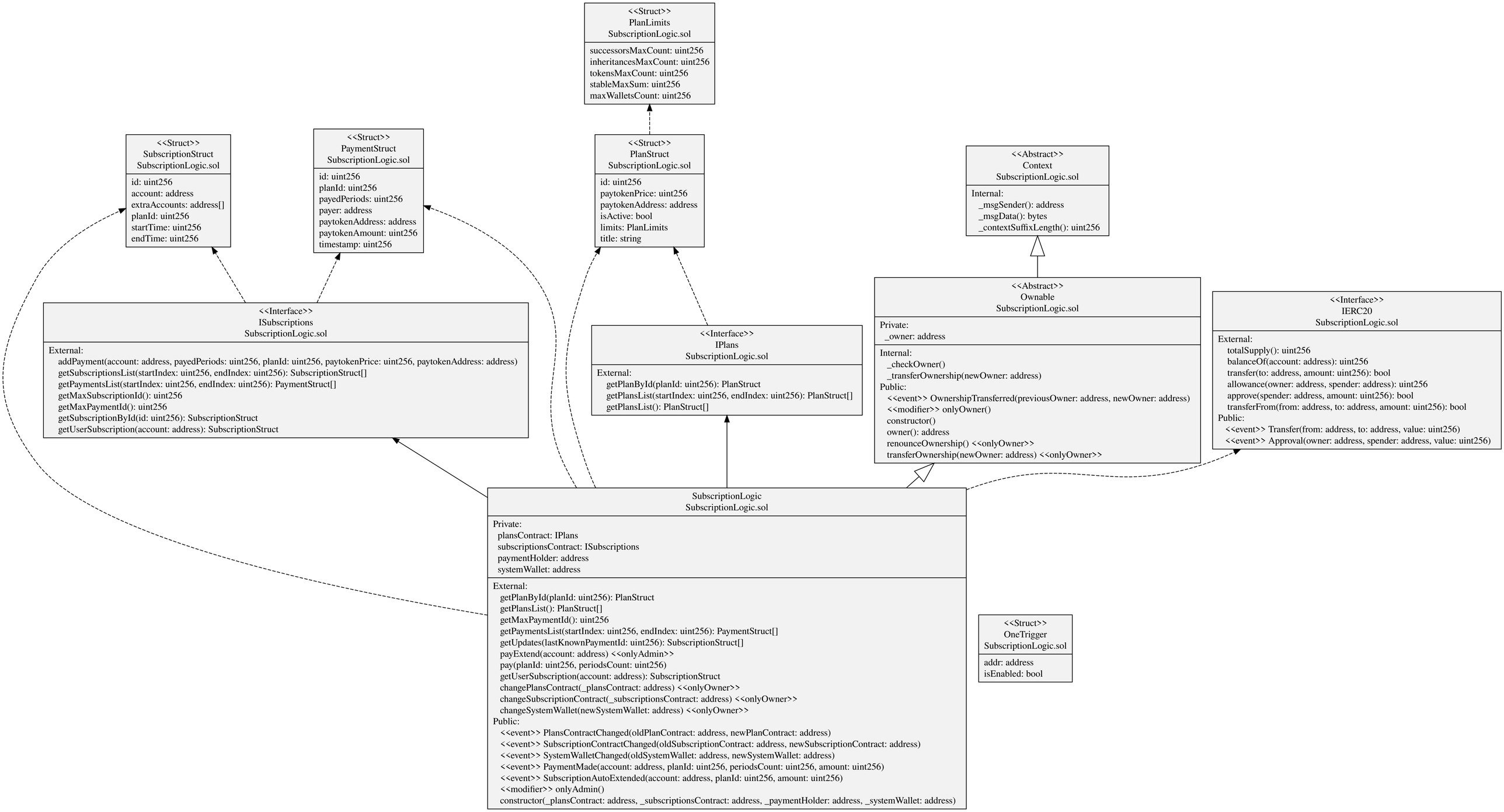
Automated Review

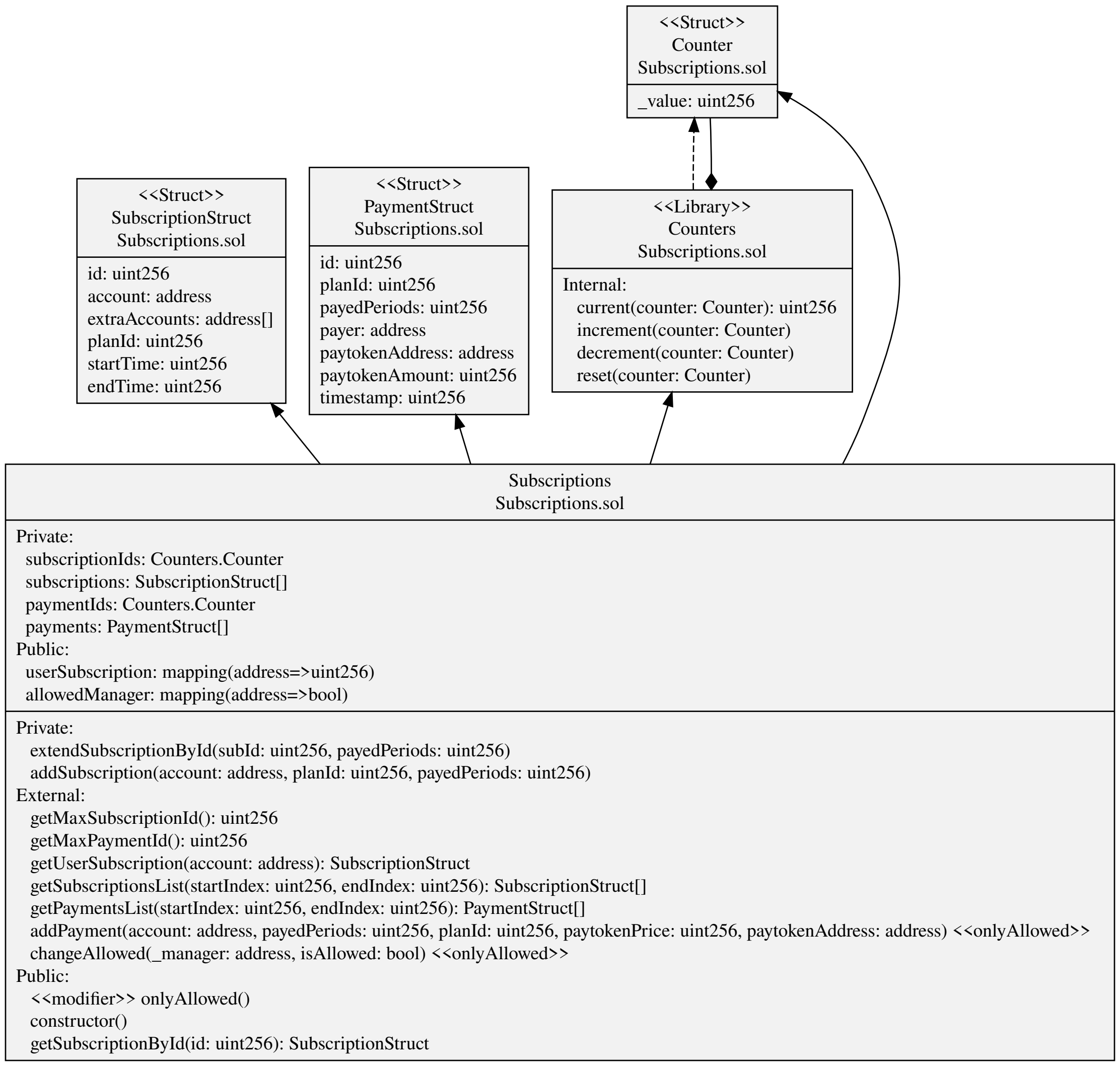

<<Interface>>
IERC165
Multipay.sol
External:
supportsInterface(interfaceId: bytes4): bool
<<Struct>>
Counter
Multipay.sol
_value: uint256
<<Enum>>
ApproveState
Multipay.sol
NotApproved: 0
PartiallyApproved: 1
Approved: 2
<<Struct>>
OneNFTallowance
Multipay.sol
id: uint256
isApproved: bool
<<Abstract>>
Context
Multipay.sol
Internal:
_msgSender(): address
_msgData(): bytes
_contextSuffixLength(): uint256
<<Interface>>
IERC20Permit
Multipay.sol
External:
permit(owner: address, spender: address, value: uint256, deadline: uint256, v: uint8, r: bytes32, s: bytes32)
nonces(owner: address): uint256
DOMAIN_SEPARATOR(): bytes32
<<Library>>
Address
Multipay.sol
Private:
_revert(returndata: bytes, errorMessage: string)
Internal:
isContract(account: address): bool
sendValue(recipient: address, amount: uint256)
functionCall(target: address, data: bytes): bytes
functionCall(target: address, data: bytes, errorMessage: string): bytes
functionCallWithValue(target: address, data: bytes, value: uint256): bytes
functionCallWithValue(target: address, data: bytes, value: uint256, errorMessage: string): bytes
functionStaticCall(target: address, data: bytes): bytes
functionStaticCall(target: address, data: bytes, errorMessage: string): bytes
functionDelegateCall(target: address, data: bytes): bytes
functionDelegateCall(target: address, data: bytes, errorMessage: string): bytes
verifyCallResultFromTarget(target: address, success: bool, returndata: bytes, errorMessage: string): bytes
verifyCallResult(success: bool, returndata: bytes, errorMessage: string): bytes
<<Interface>>
IERC20
Multipay.sol
External:
totalSupply(): uint256
balanceOf(account: address): uint256
transfer(to: address, amount: uint256): bool
allowance(owner: address, spender: address): uint256
approve(spender: address, amount: uint256): bool
transferFrom(from: address, to: address, amount: uint256): bool
Public:
<<event>> Transfer(from: address, to: address, value: uint256)
<<event>> Approval(owner: address, spender: address, value: uint256)
<<Interface>>
IERC721
Multipay.sol
External:
balanceOf(owner: address): (balance: uint256)
ownerOf(tokenId: uint256): (owner: address)
safeTransferFrom(from: address, to: address, tokenId: uint256, data: bytes)
safeTransferFrom(from: address, to: address, tokenId: uint256)
transferFrom(from: address, to: address, tokenId: uint256)
approve(to: address, tokenId: uint256)
setApprovalForAll(operator: address, approved: bool)
getApproved(tokenId: uint256): (operator: address)
isApprovedForAll(owner: address, operator: address): bool
Public:
<<event>> Transfer(from: address, to: address, tokenId: uint256)
<<event>> Approval(owner: address, approved: address, tokenId: uint256)
<<event>> ApprovalForAll(owner: address, operator: address, approved: bool)
<<Library>>
Counters
Multipay.sol
Internal:
current(counter: Counter): uint256
increment(counter: Counter)
decrement(counter: Counter)
reset(counter: Counter)
<<Struct>>
GasPayment
Multipay.sol
id: uint256
requestId: uint256
coinAmount: uint256
timestamp: uint256
payer: address
<<Struct>>
OneUserTokenInfo
Multipay.sol
token: address
balance: uint256
state: ApproveState
<<Struct>>
OneShortNFTCollectionAllowance
Multipay.sol
token: address
isApproved: bool
<<Struct>>
OneSendCollection
Multipay.sol
token: address
ids: uint256[]
successors: address[]
<<Struct>>
OneUserCollectionInfo
Multipay.sol
token: address
isApprovedForAll: bool
state: ApproveState
individualNFT: OneNFTallowance[]
<<Abstract>>
Ownable
Multipay.sol
Private:
_owner: address
Internal:
_checkOwner()
_transferOwnership(newOwner: address)
Public:
<<event>> OwnershipTransferred(previousOwner: address, newOwner: address)
<<modifier>> onlyOwner()
constructor()
owner(): address
renounceOwnership() <<onlyOwner>>
transferOwnership(newOwner: address) <<onlyOwner>>
<<Library>>
SafeERC20
Multipay.sol
Private:
_callOptionalReturn(token: IERC20, data: bytes)
_callOptionalReturnBool(token: IERC20, data: bytes): bool
Internal:
safeTransfer(token: IERC20, to: address, value: uint256)
safeTransferFrom(token: IERC20, from: address, to: address, value: uint256)
safeApprove(token: IERC20, spender: address, value: uint256)
safeIncreaseAllowance(token: IERC20, spender: address, value: uint256)
safeDecreaseAllowance(token: IERC20, spender: address, value: uint256)
forceApprove(token: IERC20, spender: address, value: uint256)
safePermit(token: IERC20Permit, owner: address, spender: address, value: uint256, deadline: uint256, v: uint8, r: bytes32, s: bytes32)
MultiPay
Multipay.sol
Private:
paymentIds: Counters.Counter
Public:
admins: mapping(address=>bool)
payments: GasPayment[]
paymentHolder: address
Internal:
sendAmountTo(account: address, totalToSend: uint256, percents: uint8[], successors: address[], token: IERC20)
External:
<<payable>> reserveGas(requestId: uint256)
getUpdates(lastKnownPaymentId: uint256): GasPayment[]
sendInheritance(account: address, tokensLimit: uint256, tokens: address[], successors: address[], percents: uint8[]) <<onlyAdmin>>
sendInheritance721(account: address, collections: OneSendCollection[]) <<onlyAdmin>>
checkInfo(account: address, tokens: address[]): OneUserTokenInfo[]
checkNftCollectionShortInfo(account: address, tokens: address[]): OneShortNFTCollectionAllowance[]
toggleAdmin(newAdmin: address, isEnabled: bool) <<onlyOwner>>
changePaymentHolder(_newPaymentHolder: address) <<onlyOwner>>
Public:
<<event>> ErrorTransferErc20(token: address, account: address, to: address, amount: uint256)
<<event>> ErrorTransferErc721(token: address, account: address, to: address, id: uint256)
<<event>> PaymentReserved(paymentId: uint256, requestId: uint256, amount: uint256, sender: address)
<<event>> InheritanceSent(account: address, tokens: address[], successors: address[], percents: uint8[])
<<event>> Inheritance721Sent(account: address, collection: OneSendCollection)
<<event>> AdminToggled(admin: address, isEnabled: bool)
<<event>> PaymentHolderChanged(oldPaymentHolder: address, newPaymentHolder: address)
<<modifier>> onlyAdmin()
constructor(_serverWallet: address, _paymentHolder: address)
getMaxPaymentId(): uint256

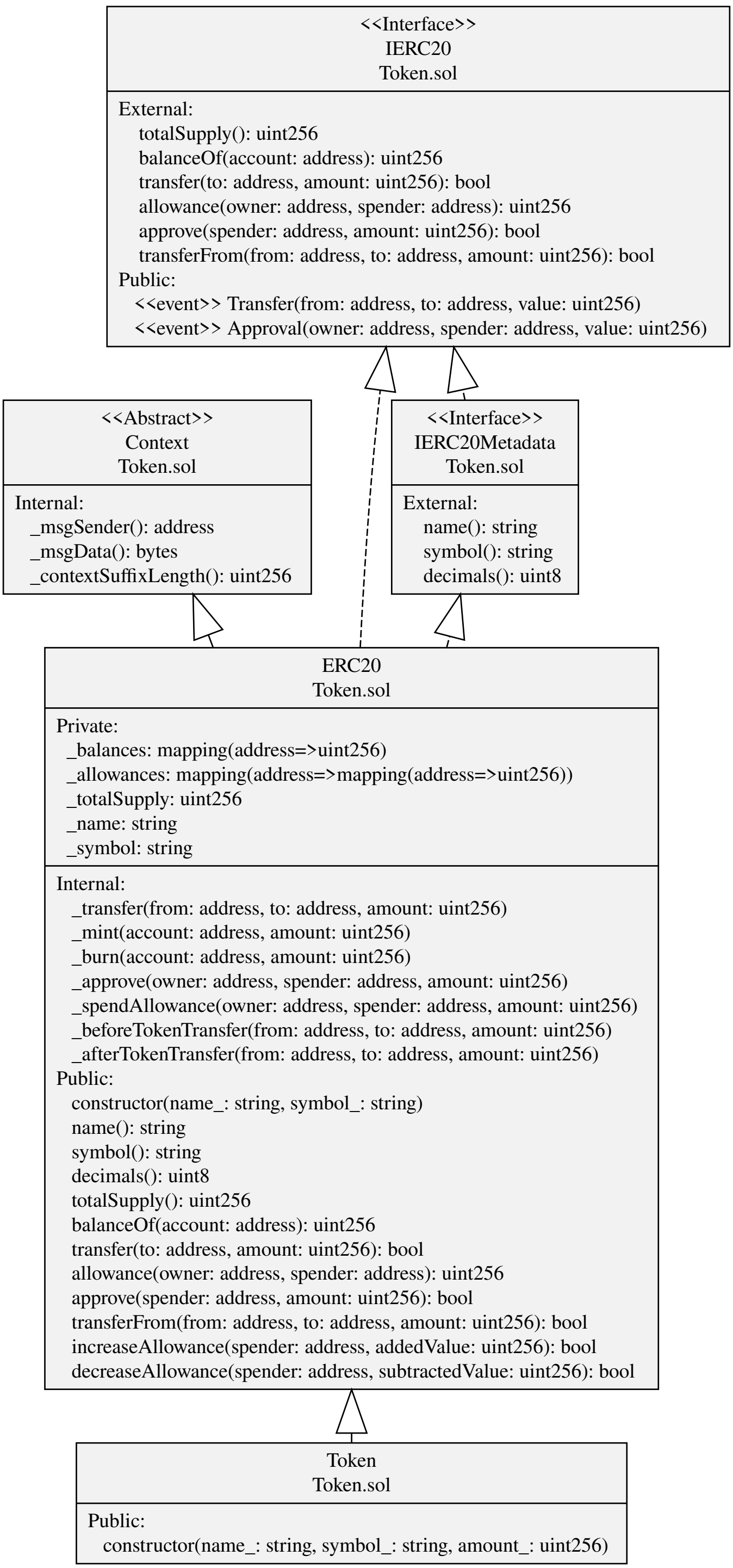

<<Interface>>
IERC165
NFT721.sol
External:
supportsInterface(interfaceId: bytes4): bool
<<Interface>>
IERC721
NFT721.sol
External:
balanceOf(owner: address): (balance: uint256)
ownerOf(tokenId: uint256): (owner: address)
safeTransferFrom(from: address, to: address, tokenId: uint256, data: bytes)
safeTransferFrom(from: address, to: address, tokenId: uint256)
transferFrom(from: address, to: address, tokenId: uint256)
approve(to: address, tokenId: uint256)
setApprovalForAll(operator: address, approved: bool)
getApproved(tokenId: uint256): (operator: address)
isApprovedForAll(owner: address, operator: address): bool
Public:
<<event>> Transfer(from: address, to: address, tokenId: uint256)
<<event>> Approval(owner: address, approved: address, tokenId: uint256)
<<event>> ApprovalForAll(owner: address, operator: address, approved: bool)
<<Library>>
SignedMath
NFT721.sol
Internal:
max(a: int256, b: int256): int256
min(a: int256, b: int256): int256
average(a: int256, b: int256): int256
abs(n: int256): uint256
<<Abstract>>
ERC165
NFT721.sol
Public:
supportsInterface(interfaceId: bytes4): bool
<<Library>>
Address
NFT721.sol
Private:
_revert(returndata: bytes, errorMessage: string)
Internal:
isContract(account: address): bool
sendValue(recipient: address, amount: uint256)
functionCall(target: address, data: bytes): bytes
functionCall(target: address, data: bytes, errorMessage: string): bytes
functionCallWithValue(target: address, data: bytes, value: uint256): bytes
functionCallWithValue(target: address, data: bytes, value: uint256, errorMessage: string): bytes
functionStaticCall(target: address, data: bytes): bytes
functionStaticCall(target: address, data: bytes, errorMessage: string): bytes
functionDelegateCall(target: address, data: bytes): bytes
functionDelegateCall(target: address, data: bytes, errorMessage: string): bytes
verifyCallResultFromTarget(target: address, success: bool, returndata: bytes, errorMessage: string): bytes
verifyCallResult(success: bool, returndata: bytes, errorMessage: string): bytes
<<Library>>
Strings
NFT721.sol
Private:
_SYMBOLS: bytes16
_ADDRESS_LENGTH: uint8
Internal:
toString(value: uint256): string
toString(value: int256): string
toHexString(value: uint256): string
toHexString(value: uint256, length: uint256): string
toHexString(addr: address): string
equal(a: string, b: string): bool
<<Interface>>
IERC721Metadata
NFT721.sol
External:
name(): string
symbol(): string
tokenURI(tokenId: uint256): string
<<Interface>>
IERC4906
NFT721.sol
Public:
<<event>> MetadataUpdate(_tokenId: uint256)
<<event>> BatchMetadataUpdate(_fromTokenId: uint256, _toTokenId: uint256)
<<Abstract>>
Context
NFT721.sol
Internal:
_msgSender(): address
_msgData(): bytes
_contextSuffixLength(): uint256
ERC721
NFT721.sol
Private:
_name: string
_symbol: string
_owners: mapping(uint256=>address)
_balances: mapping(address=>uint256)
_tokenApprovals: mapping(uint256=>address)
_operatorApprovals: mapping(address=>mapping(address=>bool))
Private:
_checkOnERC721Received(from: address, to: address, tokenId: uint256, data: bytes): bool
Internal:
_baseURI(): string
_safeTransfer(from: address, to: address, tokenId: uint256, data: bytes)
_ownerOf(tokenId: uint256): address
_exists(tokenId: uint256): bool
_isApprovedOrOwner(spender: address, tokenId: uint256): bool
_safeMint(to: address, tokenId: uint256)
_safeMint(to: address, tokenId: uint256, data: bytes)
_mint(to: address, tokenId: uint256)
_burn(tokenId: uint256)
_transfer(from: address, to: address, tokenId: uint256)
_approve(to: address, tokenId: uint256)
_setApprovalForAll(owner: address, operator: address, approved: bool)
_requireMinted(tokenId: uint256)
_beforeTokenTransfer(from: address, to: address, firstTokenId: uint256, batchSize: uint256)
_afterTokenTransfer(from: address, to: address, firstTokenId: uint256, batchSize: uint256)
__unsafe_increaseBalance(account: address, amount: uint256)
Public:
constructor(name_: string, symbol_: string)
supportsInterface(interfaceId: bytes4): bool
balanceOf(owner: address): uint256
ownerOf(tokenId: uint256): address
name(): string
symbol(): string
tokenURI(tokenId: uint256): string
approve(to: address, tokenId: uint256)
getApproved(tokenId: uint256): address
setApprovalForAll(operator: address, approved: bool)
isApprovedForAll(owner: address, operator: address): bool
transferFrom(from: address, to: address, tokenId: uint256)
safeTransferFrom(from: address, to: address, tokenId: uint256)
safeTransferFrom(from: address, to: address, tokenId: uint256, data: bytes)
<<Abstract>>
Ownable
NFT721.sol
Private:
_owner: address
Internal:
_checkOwner()
_transferOwnership(newOwner: address)
Public:
<<event>> OwnershipTransferred(previousOwner: address, newOwner: address)
<<modifier>> onlyOwner()
constructor()
owner(): address
renounceOwnership() <<onlyOwner>>
transferOwnership(newOwner: address) <<onlyOwner>>
<<Abstract>>
ERC721URIStorage
NFT721.sol
Private:
_tokenURIs: mapping(uint256=>string)
Internal:
_setTokenURI(tokenId: uint256, _tokenURI: string)
_burn(tokenId: uint256)
Public:
supportsInterface(interfaceId: bytes4): bool
tokenURI(tokenId: uint256): string
<<Enum>>
Rounding
NFT721.sol
Down: 0
Up: 1
Zero: 2
<<Library>>
Math
NFT721.sol
Internal:
max(a: uint256, b: uint256): uint256
min(a: uint256, b: uint256): uint256
average(a: uint256, b: uint256): uint256
ceilDiv(a: uint256, b: uint256): uint256
mulDiv(x: uint256, y: uint256, denominator: uint256): (result: uint256)
mulDiv(x: uint256, y: uint256, denominator: uint256, rounding: Rounding): uint256
sqrt(a: uint256): uint256
sqrt(a: uint256, rounding: Rounding): uint256
log2(value: uint256): uint256
log2(value: uint256, rounding: Rounding): uint256
log10(value: uint256): uint256
log10(value: uint256, rounding: Rounding): uint256
log256(value: uint256): uint256
log256(value: uint256, rounding: Rounding): uint256
MyNFT
NFT721.sol
Public:
constructor(_name: string, _symbol: string, _baseURI: string)
mint(to: address, tokenId: uint256, _tokenURI: string) <<onlyOwner>>
mintMany(to: address, tokenIds: uint256[], _tokenURIs: string[]) <<onlyOwner>>
<<Interface>>
IERC721Receiver
NFT721.sol
External:
onERC721Received(operator: address, from: address, tokenId: uint256, data: bytes): bytes4


<<Struct>>
PlanLimits
Plans.sol
successorsMaxCount: uint256
inheritancesMaxCount: uint256
tokensMaxCount: uint256
stableMaxSum: uint256
maxWalletsCount: uint256
<<Struct>>
Counter
Plans.sol
_value: uint256
<<Abstract>>
Context
Plans.sol
Internal:
_msgSender(): address
_msgData(): bytes
_contextSuffixLength(): uint256
<<Struct>>
PlanStruct
Plans.sol
id: uint256
paytokenPrice: uint256
paytokenAddress: address
isActive: bool
limits: PlanLimits
title: string
<<Library>>
Counters
Plans.sol
Internal:
current(counter: Counter): uint256
increment(counter: Counter)
decrement(counter: Counter)
reset(counter: Counter)
<<Abstract>>
Ownable
Plans.sol
Private:
_owner: address
Internal:
_checkOwner()
_transferOwnership(newOwner: address)
Public:
<<event>> OwnershipTransferred(previousOwner: address, newOwner: address)
<<modifier>> onlyOwner()
constructor()
owner(): address
renounceOwnership() <<onlyOwner>>
transferOwnership(newOwner: address) <<onlyOwner>>
<<Struct>>
OneTrigger
Plans.sol
addr: address
isEnabled: bool
Plans
Plans.sol
Private:
planIds: Counters.Counter
plans: PlanStruct[]
External:
getPlanId(): uint256
getPlanById(id: uint256): PlanStruct
getPlansList(): PlanStruct[]
getPlansList(startIndex: uint256, endIndex: uint256): PlanStruct[]
addPlan(title: string, paytokenAddress: address, paytokenPrice: uint256, limits: PlanLimits) <<onlyOwner>>
togglePlan(planId: uint256, isActive: bool) <<onlyOwner>>
Public:
constructor()

<<Interface>>
IERC20
StableCoin.sol
External:
totalSupply(): uint256
balanceOf(account: address): uint256
transfer(to: address, amount: uint256): bool
allowance(owner: address, spender: address): uint256
approve(spender: address, amount: uint256): bool
transferFrom(from: address, to: address, amount: uint256): bool
Public:
<<event>> Transfer(from: address, to: address, value: uint256)
<<event>> Approval(owner: address, spender: address, value: uint256)
<<Abstract>>
Context
StableCoin.sol
Internal:
_msgSender(): address
_msgData(): bytes
_contextSuffixLength(): uint256
<<Interface>>
IERC20Metadata
StableCoin.sol
External:
name(): string
symbol(): string
decimals(): uint8
ERC20
StableCoin.sol
Private:
_balances: mapping(address=>uint256)
_allowances: mapping(address=>mapping(address=>uint256))
_totalSupply: uint256
_name: string
_symbol: string
Internal:
_transfer(from: address, to: address, amount: uint256)
_mint(account: address, amount: uint256)
_burn(account: address, amount: uint256)
_approve(owner: address, spender: address, amount: uint256)
_spendAllowance(owner: address, spender: address, amount: uint256)
_beforeTokenTransfer(from: address, to: address, amount: uint256)
_afterTokenTransfer(from: address, to: address, amount: uint256)
Public:
constructor(name_: string, symbol_: string)
name(): string
symbol(): string
decimals(): uint8
totalSupply(): uint256
balanceOf(account: address): uint256
transfer(to: address, amount: uint256): bool
allowance(owner: address, spender: address): uint256
approve(spender: address, amount: uint256): bool
transferFrom(from: address, to: address, amount: uint256): bool
increaseAllowance(spender: address, addedValue: uint256): bool
decreaseAllowance(spender: address, subtractedValue: uint256): bool
StableCoin
StableCoin.sol
Public:
constructor(name_: string, symbol_: string, amount_: uint256)
decimals(): uint8


<<Struct>>
PlanLimits
SubscriptionLogic.sol
successorsMaxCount: uint256
inheritancesMaxCount: uint256
tokensMaxCount: uint256
stableMaxSum: uint256
maxWalletsCount: uint256
<<Struct>>
SubscriptionStruct
SubscriptionLogic.sol
id: uint256
account: address
extraAccounts: address[]
planId: uint256
startTime: uint256
endTime: uint256
<<Struct>>
PaymentStruct
SubscriptionLogic.sol
id: uint256
planId: uint256
payedPeriods: uint256
payer: address
paytokenAddress: address
paytokenAmount: uint256
timestamp: uint256
<<Struct>>
PlanStruct
SubscriptionLogic.sol
id: uint256
paytokenPrice: uint256
paytokenAddress: address
isActive: bool
limits: PlanLimits
title: string
<<Abstract>>
Context
SubscriptionLogic.sol
Internal:
_msgSender(): address
_msgData(): bytes
_contextSuffixLength(): uint256
<<Interface>>
ISubscriptions
SubscriptionLogic.sol
External:
addPayment(account: address, payedPeriods: uint256, planId: uint256, paytokenPrice: uint256, paytokenAddress: address)
getSubscriptionsList(startIndex: uint256, endIndex: uint256): SubscriptionStruct[]
getPaymentsList(startIndex: uint256, endIndex: uint256): PaymentStruct[]
getMaxSubscriptionId(): uint256
getMaxPaymentId(): uint256
getSubscriptionById(id: uint256): SubscriptionStruct
getUserSubscription(account: address): SubscriptionStruct
<<Interface>>
IPlans
SubscriptionLogic.sol
External:
getPlanById(planId: uint256): PlanStruct
getPlansList(startIndex: uint256, endIndex: uint256): PlanStruct[]
getPlansList(): PlanStruct[]
<<Abstract>>
Ownable
SubscriptionLogic.sol
Private:
_owner: address
Internal:
_checkOwner()
_transferOwnership(newOwner: address)
Public:
<<event>> OwnershipTransferred(previousOwner: address, newOwner: address)
<<modifier>> onlyOwner()
constructor()
owner(): address
renounceOwnership() <<onlyOwner>>
transferOwnership(newOwner: address) <<onlyOwner>>
<<Interface>>
IERC20
SubscriptionLogic.sol
External:
totalSupply(): uint256
balanceOf(account: address): uint256
transfer(to: address, amount: uint256): bool
allowance(owner: address, spender: address): uint256
approve(spender: address, amount: uint256): bool
transferFrom(from: address, to: address, amount: uint256): bool
Public:
<<event>> Transfer(from: address, to: address, value: uint256)
<<event>> Approval(owner: address, spender: address, value: uint256)
SubscriptionLogic
SubscriptionLogic.sol
Private:
plansContract: IPlans
subscriptionsContract: ISubscriptions
paymentHolder: address
systemWallet: address
External:
getPlanById(planId: uint256): PlanStruct
getPlansList(): PlanStruct[]
getMaxPaymentId(): uint256
getPaymentsList(startIndex: uint256, endIndex: uint256): PaymentStruct[]
getUpdates(lastKnownPaymentId: uint256): SubscriptionStruct[]
payExtend(account: address) <<onlyAdmin>>
pay(planId: uint256, periodsCount: uint256)
getUserSubscription(account: address): SubscriptionStruct
changePlansContract(_plansContract: address) <<onlyOwner>>
changeSubscriptionContract(_subscriptionsContract: address) <<onlyOwner>>
changeSystemWallet(newSystemWallet: address) <<onlyOwner>>
Public:
<<event>> PlansContractChanged(oldPlanContract: address, newPlanContract: address)
<<event>> SubscriptionContractChanged(oldSubscriptionContract: address, newSubscriptionContract: address)
<<event>> SystemWalletChanged(oldSystemWallet: address, newSystemWallet: address)
<<event>> PaymentMade(account: address, planId: uint256, periodsCount: uint256, amount: uint256)
<<event>> SubscriptionAutoExtended(account: address, planId: uint256, amount: uint256)
<<modifier>> onlyAdmin()
constructor(_plansContract: address, _subscriptionsContract: address, _paymentHolder: address, _systemWallet: address)
<<Struct>>
OneTrigger
SubscriptionLogic.sol
addr: address
isEnabled: bool


<<Struct>>
Counter
Subscriptions.sol
_value: uint256
<<Struct>>
SubscriptionStruct
Subscriptions.sol
id: uint256
account: address
extraAccounts: address[]
planId: uint256
startTime: uint256
endTime: uint256
<<Struct>>
PaymentStruct
Subscriptions.sol
id: uint256
planId: uint256
payedPeriods: uint256
payer: address
paytokenAddress: address
paytokenAmount: uint256
timestamp: uint256
<<Library>>
Counters
Subscriptions.sol
Internal:
current(counter: Counter): uint256
increment(counter: Counter)
decrement(counter: Counter)
reset(counter: Counter)
Subscriptions
Subscriptions.sol
Private:
subscriptionIds: Counters.Counter
subscriptions: SubscriptionStruct[]
paymentIds: Counters.Counter
payments: PaymentStruct[]
Public:
userSubscription: mapping(address=>uint256)
allowedManager: mapping(address=>bool)
Private:
extendSubscriptionById(subId: uint256, payedPeriods: uint256)
addSubscription(account: address, planId: uint256, payedPeriods: uint256)
External:
getMaxSubscriptionId(): uint256
getMaxPaymentId(): uint256
getUserSubscription(account: address): SubscriptionStruct
getSubscriptionsList(startIndex: uint256, endIndex: uint256): SubscriptionStruct[]
getPaymentsList(startIndex: uint256, endIndex: uint256): PaymentStruct[]
addPayment(account: address, payedPeriods: uint256, planId: uint256, paytokenPrice: uint256, paytokenAddress: address) <<onlyAllowed>>
changeAllowed(_manager: address, isAllowed: bool) <<onlyAllowed>>
Public:
<<modifier>> onlyAllowed()
constructor()
getSubscriptionById(id: uint256): SubscriptionStruct


<<Interface>>
IERC20
Token.sol
External:
totalSupply(): uint256
balanceOf(account: address): uint256
transfer(to: address, amount: uint256): bool
allowance(owner: address, spender: address): uint256
approve(spender: address, amount: uint256): bool
transferFrom(from: address, to: address, amount: uint256): bool
Public:
<<event>> Transfer(from: address, to: address, value: uint256)
<<event>> Approval(owner: address, spender: address, value: uint256)
<<Abstract>>
Context
Token.sol
Internal:
_msgSender(): address
_msgData(): bytes
_contextSuffixLength(): uint256
<<Interface>>
IERC20Metadata
Token.sol
External:
name(): string
symbol(): string
decimals(): uint8
ERC20
Token.sol
Private:
_balances: mapping(address=>uint256)
_allowances: mapping(address=>mapping(address=>uint256))
_totalSupply: uint256
_name: string
_symbol: string
Internal:
_transfer(from: address, to: address, amount: uint256)
_mint(account: address, amount: uint256)
_burn(account: address, amount: uint256)
_approve(owner: address, spender: address, amount: uint256)
_spendAllowance(owner: address, spender: address, amount: uint256)
_beforeTokenTransfer(from: address, to: address, amount: uint256)
_afterTokenTransfer(from: address, to: address, amount: uint256)
Public:
constructor(name_: string, symbol_: string)
name(): string
symbol(): string
decimals(): uint8
totalSupply(): uint256
balanceOf(account: address): uint256
transfer(to: address, amount: uint256): bool
allowance(owner: address, spender: address): uint256
approve(spender: address, amount: uint256): bool
transferFrom(from: address, to: address, amount: uint256): bool
increaseAllowance(spender: address, addedValue: uint256): bool
decreaseAllowance(spender: address, subtractedValue: uint256): bool
Token
Token.sol
Public:
constructor(name_: string, symbol_: string, amount_: uint256)