
ANICANA Technical Documents

ANICANA PJ

May 01, 2024

RELEASE INFORMATION FOR EACH PLATFORM FUNCTION:

1	Release Information List	1
2	LEVICA-appProduction	3
2.1	release information	3
3	LEVICA-appArk.one	5
3.1	release information	5
4	LEVICA APIProduction	7
4.1	release information	8
5	LEVICA APIArk.one	9
5.1	release information	10
6	ARCANA Generation APIProduction	11
6.1	release information	11
7	ARCANA Generation APIArk.one	13
7.1	release information	13
8	LEVIAS IDProduction	15
8.1	release information	15
9	LEVIAS IDArk.one	17
9.1	release information	17
10	OctillionProduction	19
10.1	release information	19
11	OctillionArk.one	21
11.1	release information	21
12	API ServerProduction	23
12.1	release information	23
13	API ServerArk.one	25
13.1	release information	25
14	AdSquareProduction	27
14.1	release information	27

15 AdSquareArk.one	29
15.1 release information	29
16 What is ANICANA Chain?	31
16.1 Consortium-Based Private Chain	31
17 System Requirements	33
18 Consensus Algorithm	35
19 How to Join ANICANA	37
20 Quick Start for Publishers	39
20.1 Initial Setup	39
20.2 Content Development	40
20.3 Operations	41
21 Knights of the Round Table	43
21.1 Permission-type Nodes Granting Approval for Validators	43
21.2 Basic Structure	43
22 Roles in the Knights of the Round Table	45
22.1 Roles	45
22.2 Receipt of ANM by Knights & Queen	46
22.3 Queen's Election	46
22.4 Vote of No Confidence against the Queen	46
22.5 Opening of Validator (Candidate) Nodes	47
23 Mechanism of ARCANA Generation	49
23.1 ARCANA Generation	51
23.2 Storage of ARCANA Generation Information (IPFS)	51
23.3 Storage of ARCANA Generation Information Index (Contract)	51
24 ARCANA Life Cycle	53
24.1 ARCANA Life Cycle	53
25 ANICANA Life Cycle	55
25.1 Development Engineers / System Vendors, etc.	56
25.2 Content Owners / Publishers (Validators)	56
25.3 Users / Content Users	56
25.4 Secondary Marketplace / Market Operators	56
25.5 Extractors / Disassembling Engineers	56
26 Validator Setup Procedure	57
27 System Configuration	59
27.1 Configuration Diagram	59
28 ANICANA Wallet Registration	61
28.1 Wallet Registration Procedure	61
29 Preparing Your AWS Account	69
30 Validator Setup	71
31 Apply as a Validator Candidate	73

32	Generating Square Keys	75
33	Content Development Overview	77
33.1	Implementation Flow	77
34	Ark.one Environment Information	79
34.1	Environment Information	79
34.2	Contract Addresses	80
34.3	Contract ABI	80
34.4	Interfaces	81
34.5	Libraries	81
34.6	ANICANA Portal Site	81
34.7	Call ARCANA Generation Page Script	81
34.8	Check Status	82
34.9	Login Script	82
34.10	LEVICA	82
34.11	IPFS	82
35	Production Environment Information	83
35.1	Environment Information	83
35.2	Contract Addresses	84
35.3	Contract ABI	84
35.4	Interfaces	85
35.5	Libraries	85
35.6	ANICANA Portal Site	85
35.7	ARCANA Generation Page Invocation Script	85
35.8	check status	86
35.9	Login Script	86
35.10	LEVICA	86
35.11	IPFS	86
36	User Wallet Retrieval	87
36.1	User Registration Flow	87
37	Wallet Connection	89
37.1	API Specification	89
38	ARCANA Generation Process	91
38.1	Integration Flow with ARCANA Generation Page (API)	91
38.2	ARCANA Generation on the Ark.one	92
39	ARCANA Generation API	93
39.1	API Specifications	93
40	Get a List of Owned EGGs	97
41	Signature Generation Procedure	99
41.1	Creation of Signature Data for ARCANA Generation	99
41.2	Creation of Signature Data for PERSONA Distribution	99
41.3	Libraries	100
42	LEVICA Payment	101
42.1	API URL Format	101
42.2	Environment Information	101
42.3	Request Authentication	101
42.4	Integration Flow with LEVICA	103

42.5	Main APIs for Integration	103
42.6	Testing in the Staging Environment	108
43	PERSONA Overview	109
43.1	PERSONA Growth / Object Absorption Contract	109
43.2	DrawChain / Contract to Retrieve Specific Data	109
44	PERSONA Implementation Procedure	111
45	Absorbing	113
45.1	Overview Diagram	114
45.2	Setting Absorbing Conditions	115
46	Implementation of DrawChain	119
46.1	Overview	119
46.2	Setting Up DrawChain	120
46.3	Other DrawChain Functions	121
46.4	Executing DrawChain	123
46.5	Implemented IDrawChainAuthorizers	124
46.6	Contract to Limit the Ability Values of PERSONAs that Can Draw (DrawAbilityLimiter.sol)	124
46.7	Contract to Limit PERSONA Categories that Can Draw (DrawPersonaCategoryLimiter.sol)	125
46.8	Contract to Limit the Number of Draws (DrawQuantityLimiter.sol)	125
46.9	Contract to Limit the Caller of draw() to Subscribers of the Square Key Associated with DrawChain (DrawFollowerLimiter.sol)	125
46.10	Contract to Limit the Number of draw() Calls by the Same PERSONA (DrawCountLimiter.sol)	126
46.11	Contract to Limit draw() Calls to Specific PERSONAs (DrawPersonaLimiter.sol)	126
47	Generating and Distributing PERSONA	127
47.1	Overview Diagram	127
47.2	Generating PERSONA	127
47.3	Absorbing	128
47.4	PERSONA Contract	128
48	Using PERSONA as a User	133
48.1	Absorbing	133
48.2	Executing DrawChain	136
49	ARCANA Generation Information	139
49.1	ARCANA Generation Information	139
50	ANICANA API	141
50.1	Detail API	141
50.2	Ipfs Upload API	141
51	Affiliate Functionality	143
51.1	Implementation Flow - Issuance of Referral Code	143
51.2	Implementation Flow - Use of Referral Code	146
52	Requesting Matrix Development	147
52.1	Matrix Development	147
52.2	Matrix Construction Steps	147
52.3	Matrix Standards	148
52.4	Matrix Templates	148
53	Generating EGGs	149
53.1	Generation Process	149

54	Checking EGG Inventory	151
55	Uploading to IPFS	153
55.1	Environment Information	153
56	Obtaining ANM (ANIMA)	155
56.1	What is ANIMA (ANM)?	155
56.2	Instances Where ANIMA is Required as Gas	155
56.3	ANIMA Generation Logic	155
57	Validator Management Interface	157
57.1	Obtaining a Private Key	157
57.2	DASHBOARD	157
57.3	PROFILE	158
57.4	EGG	160
57.5	KNIGHT	160
57.6	QUEEN	163
58	LEVICA Merchant Management Screen	165
58.1	Payment History Screen	165
58.2	All Billing History Screen	166
58.3	Reuse Settings Screen	166
58.4	About Reuse Functionality	166
59	Interface Specifications	169
59.1	MATRIX Standard	169
59.2	Other Smart Contract Interfaces	169
60	Gene Calculation	171
60.1	Gene Overview	171
60.2	Data Structure of Gene Information	171
60.3	Gene Calculation for EGG Generation from SHARD	171
60.4	About Mutation	172
61	ARCANA Extraction	173
61.1	Duration of ARCANA Extraction	173
61.2	Number of ARCANA SHARDs Obtained in ARCANA Extraction	174
61.3	Steps for ARCANA Extraction	174
62	ARCANA Attribute Value Calculation	175
62.1	Lottery Probability of Green Stars	175
62.2	Lottery Probability of Elements	175
63	Bloodline	177
63.1	Overview	177
63.2	Information Retrieval	177
63.3	List of Bloodlines	178
64	Tenebrae Overview	181
64.1	Overview	181
64.2	Issuance of Tenebrae	181
64.3	Equipping Tenebrae	181
64.4	Activation of Tenebrae	181
64.5	Tenebrae Flow	185

65 Implementation of Tenebrae	187
65.1 Operator (RECIPE Owner)	188
65.2 Manufacturer	192
65.3 Consumer (TENEBRAE Token Owner)	192
65.4 Publisher	193
66 Advanced Security Settings for Wallet Connection	195
66.1 One-Time Token	196
67 Update History	197

RELEASE INFORMATION LIST

Please see each page for details of the release.

No	Date Updated	Contents
1.		<i>LEVICA-appProduction</i>
2.		<i>LEVICA-appArk.one</i>
3.	2024/05/01	<i>LEVICA APIProduction</i>
4.	2024/05/01	<i>LEVICA APIArk.one</i>
5.		<i>ARCANA Generation APIProduction</i>
6.		<i>ARCANA Generation APIArk.one</i>
7.	2024/05/01	<i>LEVIAS IDProduction</i>
8.	2024/05/01	<i>LEVIAS IDArk.one</i>
9.		<i>OctillionProduction</i>
10.		<i>OctillionArk.one</i>
11.	2024/03/10	<i>API ServerProduction</i>
12.	2024/03/10	<i>API ServerArk.one</i>
13.		<i>AdSquareProduction</i>
14.		<i>AdSquareArk.one</i>

LEVICA-APPPRODUCTION

2.1 release information

No	Version	Release Date	Release Notes
1.	1.0.8		Latest version as of 2024/03/15

LEVICA-APPARK.ONE

3.1 release information

No	Version	Release Date	Release Notes
1.	1.0.8		Latest version as of 2024/03/15

LEVICA APIPRODUCTION

4.1 release information

No	Version	Release Date	Release Notes
3.	1.3.1	2024/05/01	<p><New Feature></p> <p>Create an API to reuse all past payments.</p> <p>Re-enable the API, all past payments for that merchant will be treated as reused, and the levica for all payments will be sent to the specified wallet address.</p> <p>Create an API to reuse all past payments made by the merchant.</p>
2.	1.3.0	2024/05/01	<p><New Feature></p> <p>LEVICA merchant management screen will be released.</p> <p>User payment history for merchants and payment information from LEVICA to merchants can be viewed.</p> <p>Also, a portion of the user payment is not reimbursed by bank transfer,</p> <p>We will also add the ability to set up the automatic transfer of a portion of a user's payment to another LEVICA account for re-use.</p>

LEVICA APIARK.ONE

5.1 release information

No	Version	Release Date	Release Notes
3.	1.3.1	2024/05/01	<p><New Feature></p> <p>Create an API to reuse all past payments.</p> <p>Re-enable the API, all past payments for that merchant will be treated as reused, and the levica for all payments will be sent to the specified wallet address.</p> <p>Create an API to reuse all past payments made by the merchant.</p>
2.	1.3.0	2024/03/18	<p><New Feature></p> <p>LEVICA merchant management screen will be released.</p> <p>User payment history for merchants and payment information from LEVICA to merchants can be viewed.</p> <p>Also, a portion of the user payment is not reimbursed by bank transfer,</p> <p>We will also add the ability to set up the automatic transfer of a portion of a user's payment to another LEVICA account for re-use.</p>

ARCANA GENERATION APIPRODUCTION

6.1 release information

No	Version	Release Date	Release Notes
1.	1.28.6		Latest version as of 2024/03/15

ARCANA GENERATION APIARK.ONE

7.1 release information

No	Version	Release Date	Release Notes
1.	1.28.6		Latest version as of 2024/03/15

LEVIAS IDPRODUCTION

8.1 release information

No	Version	Release Date	Release Notes
2.	2.0.0	2024/05/01	levias id login page change https://leviasid.anicana.org/login/idms
1.	1.3.10		Latest version as of 2024/03/15

LEVIAS IDARK.ONE**9.1 release information**

No	Version	Release Date	Release Notes
2.	2.0.0	2024/05/01	levias id login page change https://leviasid.anicana.org/login/idms
1.	1.3.10		Latest version as of 2024/03/15

OCTILLIONPRODUCTION

10.1 release information

No	Version	Release Date	Release Notes
1.	1.2.5		Latest version as of 2024/03/15

OCTILLIONARK.ONE

11.1 release information

No	Version	Release Date	Release Notes
1.	1.2.5		Latest version as of 2024/03/15

API SERVERPRODUCTION

12.1 release information

No	Version	Release Date	Release Notes
2.	1.21.1	2024/03/09	Suppress transaction failures caused by exceeding gas limits.
1.	1.21.0	2024/03/02	approve/transfer Improved api throughput.

API SERVERARK.ONE

13.1 release information

No	Version	Release Date	Release Notes
2.	1.21.1	2024/03/09	Suppress transaction failures caused by exceeding gas limits.
1.	1.21.0	2024/03/02	approve/transfer Improved api throughput.

ADSQUAREPRODUCTION

14.1 release information

No	Version	Release Date	Release Notes
1.			Not released as of 2024/03/15

ADSQUAREARK.ONE

15.1 release information

No	Version	Release Date	Release Notes
1.			Not released as of 2024/03/15

WHAT IS ANICANA CHAIN?

16.1 Consortium-Based Private Chain

ANICANA is a Quorum-based private chain, on which Validators constitute a decentralized network. Various token issues based on people's experiences, and smart contracts for token transfers adhere to standards like ERC-1155, ERC-721, ERC-20, and more. An overview of the main use cases is as follows:

16.1.1 1. Become a Network Validator

With implementing the ANICANA package, you can become a participating node (Validator) after being approved by a permission-type node (Knight). The process is designed to be operational even with minimal blockchain expertise.

16.1.2 2. Send Transactions to the Network

Nodes have a JSON-RPC interface compatible with Ethereum series (EVM), allowing connections from existing Ethereum-related tools like Metamask and Remix IDE. Additionally, basic operations can be performed from the ANICANA-exclusive wallet.

SYSTEM REQUIREMENTS

Basic Specifications

OS	Linux/UNIX-based OS
CPU	4 cores
Memory	8GB
Disk	100GB

Middleware

No	Item	Version	Purpose
1.	git	2.24.3	Fetching open-source code from platforms like GitHub
2.	gcc	4.0.0 or above	Building Quorum
3.	go	1.14.4 or above	Building Quorum
4.	geth	1.9.7-stable	Running Quorum nodes
5.	istanbul-tools	v1.0.3	Generating istanbul private chain genesis parameters
6.	node	v10.20.0 or above	Running Quorum node applications (scripts)
7.	npm	6.14.5 or above	Installing Node packages
8.	solc	2.6.10 or above	Solidity compiler
9.	python3.0	3.0 or above	Contract deployment and script execution

CONSENSUS ALGORITHM

ANICANA's consensus algorithm adopts IBFT (Istanbul Byzantine Fault Tolerance). The Blocks generated by IBFT are signed by the block's proposing node and multiple validators, therefore have very high tamper resistance because it is necessary to know all of the private keys in order to tamper with them.

HOW TO JOIN ANICANA

To participate in ANICANA, you will need to go through the following three main steps:

1. Validator Initial Setup
2. Content Development
3. System Operation

Each chapter of this document explains the procedures for each of these steps.

QUICK START FOR PUBLISHERS

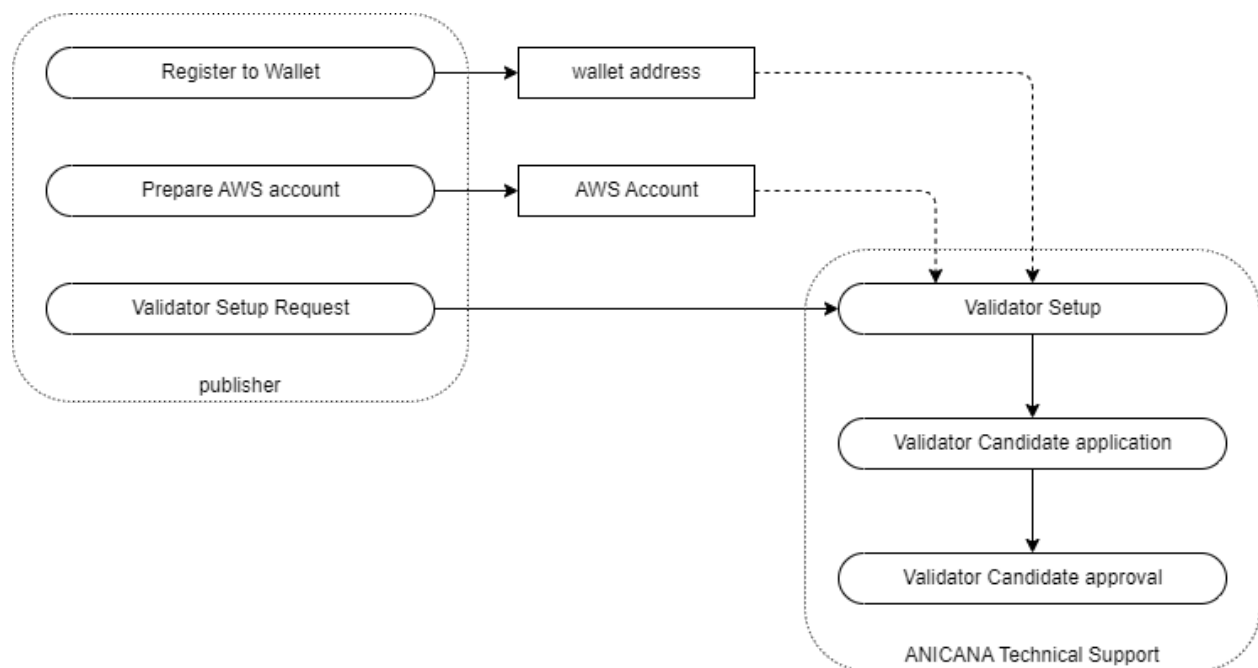
This guide introduces the necessary steps to participate as a Validator in the ANICANA network and perform ARCANA generation.

Items marked with are to be carried out by the publisher, while the ANICANA technical support team handles the rest.

20.1 Initial Setup

20.1.1 Steps to Join the ANICANA Network

1. Perform registration in the wallet to obtain a wallet address.
2. Prepare an AWS account.
3. Set up the Validator.



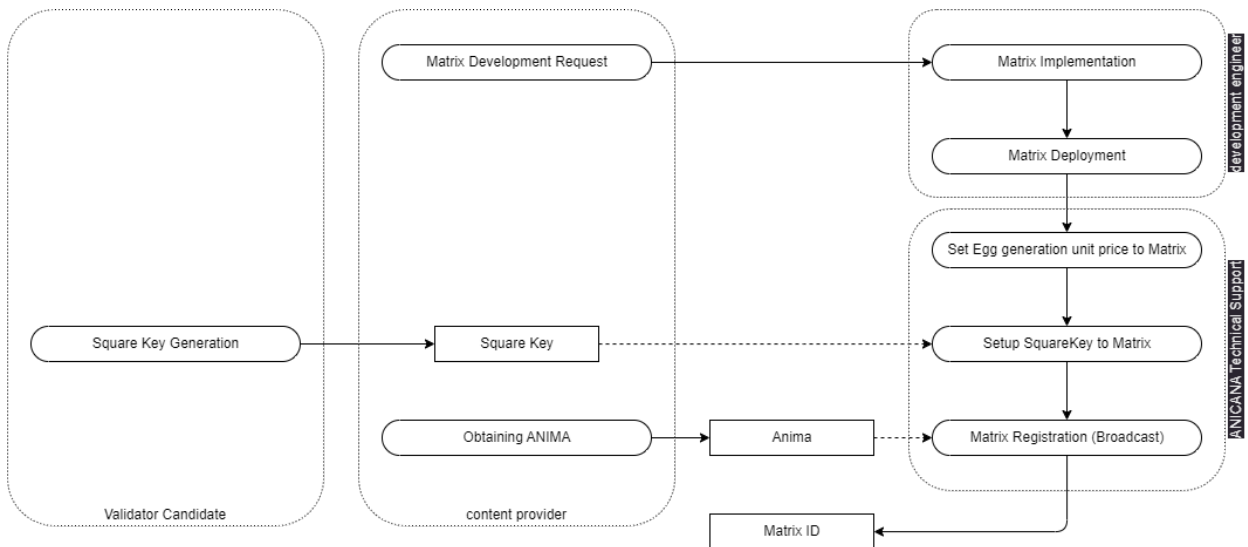
Reference Pages

Content	Reference Page
Obtain Validator Blockchain Address in ANICANA Wallet	<i>Register in ANICANA Wallet</i>
Prepare AWS (Amazon Web Services) Account	<i>Prepare AWS Account</i>
Set Up Validator Components on AWS	<i>Validator Setup</i>
Apply as a Participating Node in Knights of the Round Table	<i>Apply as Validator Candidate</i>
About Knights of the Round Table	<i>Knights of the Round Table; Roles in Knights of the Round Table</i>

20.2 Content Development

20.2.1 Steps to Build Matrix

1. Implement and deploy the Matrix.
2. Generate the SquareKey.
3. Obtain ANIMA.
4. Set pricing for Egg generation in Matrix, configure SquareKey, and register.
5. Obtain MatrixID through the above steps.



Reference Pages

Content	Reference Page
Obtain Square Key for EGG Management	<i>Generate Square Key</i>
Request Development of Matrix for EGG Generation	<i>Request Matrix Development</i>
Obtain ANM (ANIMA) for Registering Matrix	<i>Get ANM (ANIMA)</i>
Explanation of Shard, EGG, ANIMA, and ARCANA	<i>ARCANA life cycle; ANICANA life cycle</i>

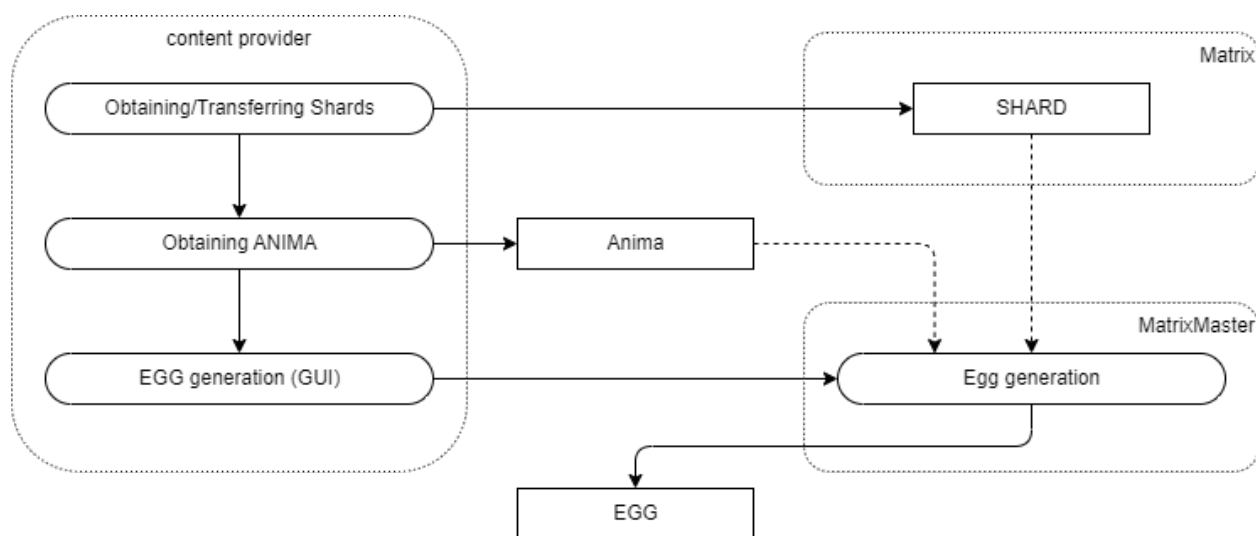
Obtaining ANIMA

ANIMA may be provided by the Technical Support Team.

20.3 Operations

20.3.1 Steps to Generate EGGs

1. Obtain Shard and transfer it to Matrix.
2. Obtain ANIMA.
3. Generate Egg via Validator management panel.



Reference Pages

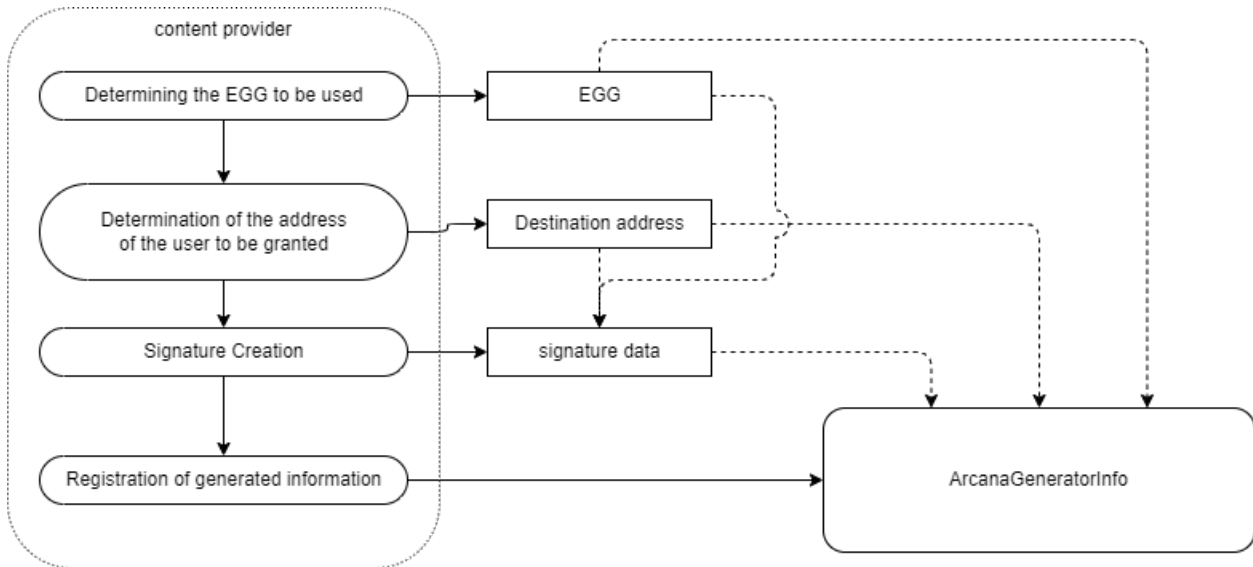
Content	Reference Page
Collect Shard	Decompose ARCANA
Obtain ANM (ANIMA) for EGG Generation	Get ANM (ANIMA)
Perform EGG Generation	Generate EGGs
Explanation of Shard, EGG, ANIMA, and ARCANA	ARCANA life cycle; ANICANA life cycle

Obtaining Shard and ANIMA

Shard and ANIMA may be provided by our technical support team.

20.3.2 Steps to Generate ARCANA

1. Determine the EGG to be used for ARCANA generation.
2. Determine the wallet address of the user to whom ARCANA will be granted.
3. Create signature data.
4. Generate ARCANA via the generation API.



Reference Pages

Content	Reference Page
Obtain Information of EGGs for ARCANA Generation	Get Owned EGGs
Link User Information and Wallet Address	Get User Wallet; Wallet Connection
Create Signature Data for ARCANA Generation	Generate Signature Steps
Interact with ARCANA Generation API	ARCANA Generation Steps; ARCANA Generation API
Mechanism of ARCANA Generation	ARCANA Generation Mechanism

KNIGHTS OF THE ROUND TABLE

21.1 Permission-type Nodes Granting Approval for Validators

The Knights of the Round Table is a group of permission-type nodes with the authority to grant approval for Validators who wish to join the ANICANA network.

Validators must receive network participation approval from at least one Knight.
(Excluding the Initial Validator)

21.2 Basic Structure

Knights of the Round Table has the roles of Queen, Knights, who can participate in blockchain consensus, and Pawns, who do not participate in consensus.

The roles may change, and the nodes participating in the consensus change accordingly.

Below are the basic composition and role selection methods.

1. The Knights of the Round Table consist of a fixed number of 13 members.
2. The 13 members consist of one Queen and 12 Knights.
3. Knights are assigned a number from 1 to 12 by the Queen.
4. Knights cannot refuse the appointment by the Queen.
5. Validators must receive network participation approval from at least one Knight. Approved Validators are linked to the number of the approving Knight and recorded on the blockchain.
6. The Queen has the authority to appoint and dismiss the 12 Knights.
7. The Queen automatically receives a certain percentage of ANM from Validators.
8. The Queen's term lasts for 2 years (specified start and end dates).
9. When the Queen's term expires, the next Queen is elected by the votes of the 12 Knights.
10. Only Validators have the right to become Knights.
11. Knights automatically receive a certain percentage of ANM from the Gas (ANM) paid by the Validators (Publishers) they have approved, based on the number assigned to the Knights.

ROLES IN THE KNIGHTS OF THE ROUND TABLE

22.1 Roles

Each Validator holds one of the following roles or attributes. For details on the authority and appointment procedures for each role, refer to the White Paper. An overview is provided below.

Role	Maximum Count	Authority	Participation in Consensus	EKG Generation
Queen	1	Nomination of Knights	Yes	Allowed
Knight	12	Approval of Validator Participation	Yes	Allowed
Pawns (or Candidates)	209	N/A	No	Allowed
Unapproved Nodes	Unlimited	N/A	No	Not Allowed

- Queen

The Queen holds the authority to nominate and dismiss Knights for Validators participating in ANICANA. The Queen is elected by a vote of 12 Knights, with a term limit of 2 years and a maximum of 2 re-elections.

- Knights

Knights have the authority to approve the participation of Validators who want to join ANICANA.

Validators appointed as Knights are assigned a Knight number.

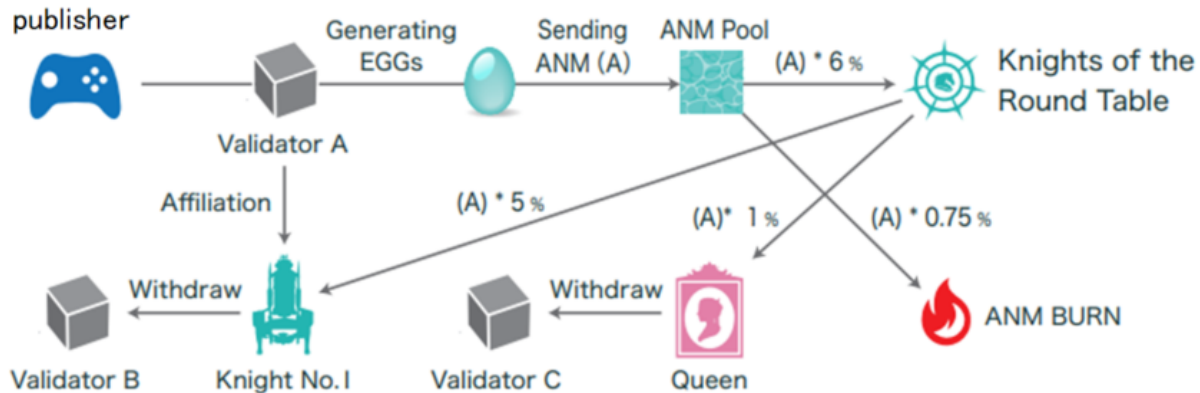
Validators cannot reject an appointment by the Queen. When a new Queen is determined, Knights are newly appointed by the Queen.

- Pawns

Pawns are nodes that has joined ANICANA or are candidates for joining ANICANA by being approved by one Knight. New Validators who join are assigned the number of the Knight who approved their participation.

22.2 Receipt of ANM by Knights & Queen

The Queen automatically receives a certain percentage of ANM from Validators. Knights automatically receive a certain percentage of ANM from the Gas (ANM) paid by the Validators (Publishers) they have approved, based on their assigned number.



22.3 Queen's Election

Only Queen and Knights, totaling 13 nodes, can participate in the blockchain consensus. These nodes propose and approve new blocks. Roles may change, and the nodes participating in the consensus change accordingly.

- A new Queen is elected from the 12 Knights.
- There is no concept of candidacy in the Queen's election, and Knights cannot vote for themselves.
- A Knight is elected as Queen by obtaining a minimum of 6 votes.
- If the minimum vote requirement is not met, the election will continue until someone achieves the minimum required votes.
- During the absence of the Queen, the supply of ANM to Knights of the Round Table is temporarily suspended, and the expected ANM will disappear. Therefore, the election of the Queen must be determined early.

22.4 Vote of No Confidence against the Queen

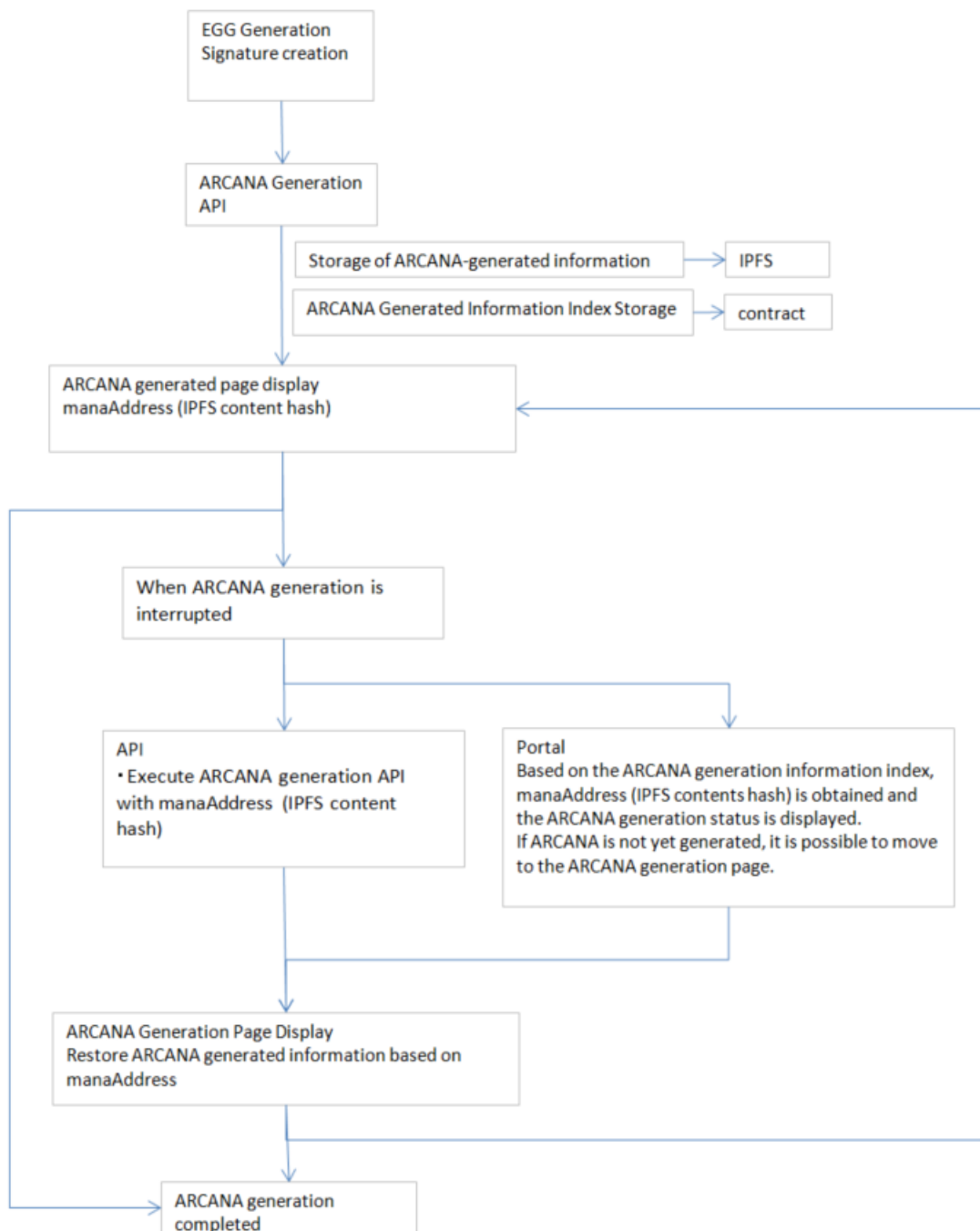
- With the agreement of three or more Knights, a vote of no confidence in the Queen can be submitted.
- When a vote of no confidence is submitted, Knights must vote either "confidence" or "no-confidence."
- The Queen can be dismissed with the votes of 9 Knights of no confidence.
- A Validator who is removed from office due to a vote of no confidence becomes a regular Validator.
- Even if a Knight is reappointed by the new Queen, there is no guarantee that existing Knights will be reappointed.
- If a Knight is reappointed, they will receive ANM linked to the number (seat) of the reappointed Knight's number, even if the Knight's number (seat) was previously different.

22.5 Opening of Validator (Candidate) Nodes

The total number of Queen, Knights, and Validators has a fixed upper limit, which is a maximum of 222. They are sequentially opened according to a pre-designed schedule before reaching the maximum number.

Release Year	Quantity	Cumulative
2022	30	30
2023	100	130
2024	50	180
2025	25	205
2026	12	217
2027	5	222

MECHANISM OF ARCANA GENERATION



23.1 ARCANA Generation

Users can generate ARCANA using the result values of playing content.

ARCANA can be used for nurturing tokens (PERSONA) provided by content owners or sold in the marketplace.

By incorporating ARCANA generation functionality, ARCANA generation can be done as a reward for using content.

For more details on generation, refer [here](#).

For information on obtaining generation information, refer [here](#).

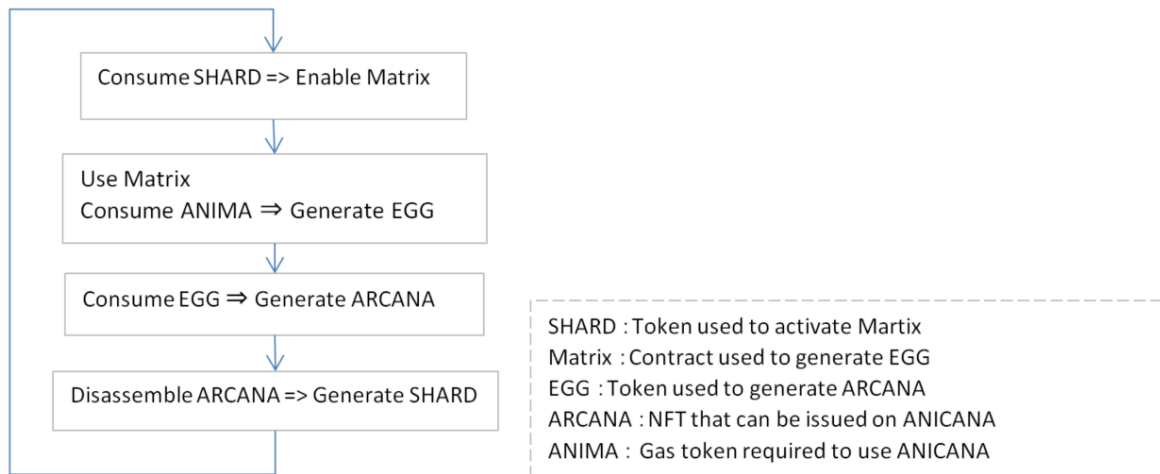
23.2 Storage of ARCANA Generation Information (IPFS)

ARCANA generation information is stored on IPFS. Based on this information, interrupted ARCANA generation can be resumed.

23.3 Storage of ARCANA Generation Information Index (Contract)

Storage of indexes to obtain IPFS mana information.

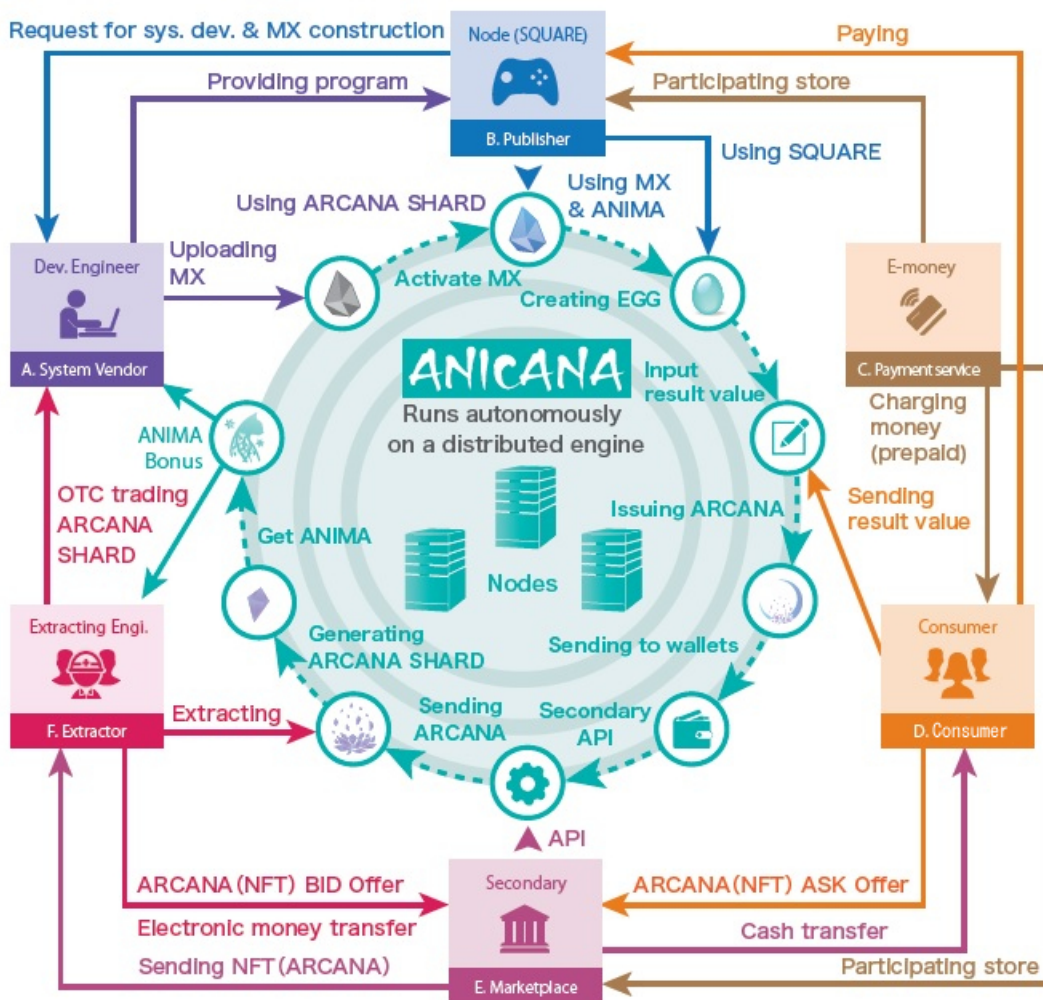
ARCANA LIFE CYCLE



24.1 ARCANA Life Cycle

1. The EGG tokens, which serve as the basis for generating ARCANA tokens, are generated by the Matrix contract. To activate this Matrix contract, SHARDs are consumed.
2. Using the activated Matrix contract to generate EGG tokens. To generate EGG tokens, ANIMA is consumed.
3. Consuming EGGs to generate ARCANA.
4. By disassembling the generated ARCANA, SHARDs can be generated.
5. The generated SHARDs are used in step 1.

ANICANA LIFE CYCLE



1. ARCANA SHARD Authentication code required to activate the smart contract (MX)
2. MX Smart contract specification
3. EGG Token's substrate contract
4. SQUARE Key Key object required for smart contract processing (creating EGG)
5. ANIMA Gas Token
6. ARCANA NFT that can be issued from EGG

25.1 Development Engineers / System Vendors, etc.

They mainly provide programs related to the development of MATRIX (hereinafter referred to as “MATRIX”), a smart contract standard that runs on ANICANA, and various authentication for each Validator (content owner / publisher). Development engineers design smart contracts based on development contracts for MATRIX from each content owner (publisher) and earn income by broadcasting MATRIX using ARCANA SHARDS. In addition, in the circular flow structure, they can receive ANIMA generated by specified tasks.

25.2 Content Owners / Publishers (Validators)

Publishers can develop content by linking their services with ANICANA. Using the environment (interface call) provided by the publishers, it is possible to generate ARCANA (NFT) on ANICANA using the results of the prepared content. Publishers can also provide and sell their own unique tokens (PERSONA) to users and can earn revenue from the sales of contents and PERSONAs, etc.

25.3 Users / Content Users

Users can purchase prepaid electronic money, etc., and charge for content. Users who use the content can generate their own ARCANA (NFT) on ANICANA by sending the result value of the content to the Egg (environment) on ANICANA. Additionally, they can nurture PERSONA by obtaining tokens (PERSONA) provided by content owners, and they can earn revenue by selling ARCANA and PERSONA.

25.4 Secondary Marketplace / Market Operators

It is a service where NFTs and item data related to content on ANICANA can be bought and sold, mainly used for trading ARCANAs and PERSONAs. When the offers of the seller and the buyer are matched (contracted), the marketplace receives the payment and tokens from the buyer, confirms that the payment has been made, and remits the payment and tokens to the seller.

25.5 Extractors / Disassembling Engineers

They receiving the authentication code “ARCANA SHARD” inherent in ARCANAs, they will become buyers of ARCANAs that are sold on the secondary marketplace. Since there is demand for ARCANA SHARD among development engineers, disassembling engineers can earn by selling the those ARCANA SHARDS to development engineers through business contracts. Also, similar to development engineers, they can receive ANIMA generated by specified tasks in the circular flow structure.

VALIDATOR SETUP PROCEDURE

The initial setup of the Validator follows the steps outlined below. The ANICANA technical support team will assist with the setup.

Step	Details
Register with the Wallet	Obtain a blockchain address for the Validator using the ANICANA wallet.
Prepare AWS Account	Prepare an account for AWS (Amazon Web Services).
Validator Setup	Set up various elements of the Validator on AWS.
Apply to Be a Validator Candidate	Get approval as a participating node from the Knights of the Round Table.
Generate Square Key	Obtain the Square Key required for EGG management.

Sure, here's the translation:

SYSTEM CONFIGURATION

A single Validator is composed of the following components:

Component	Description
Quorum Node	The core ANICANA blockchain node
Key Management System	System for managing Validator's private keys
Validator Management UI	UI for managing Validator nodes and EGGs
Explorer	UI for viewing transactions on the ANICANA chain

27.1 Configuration Diagram

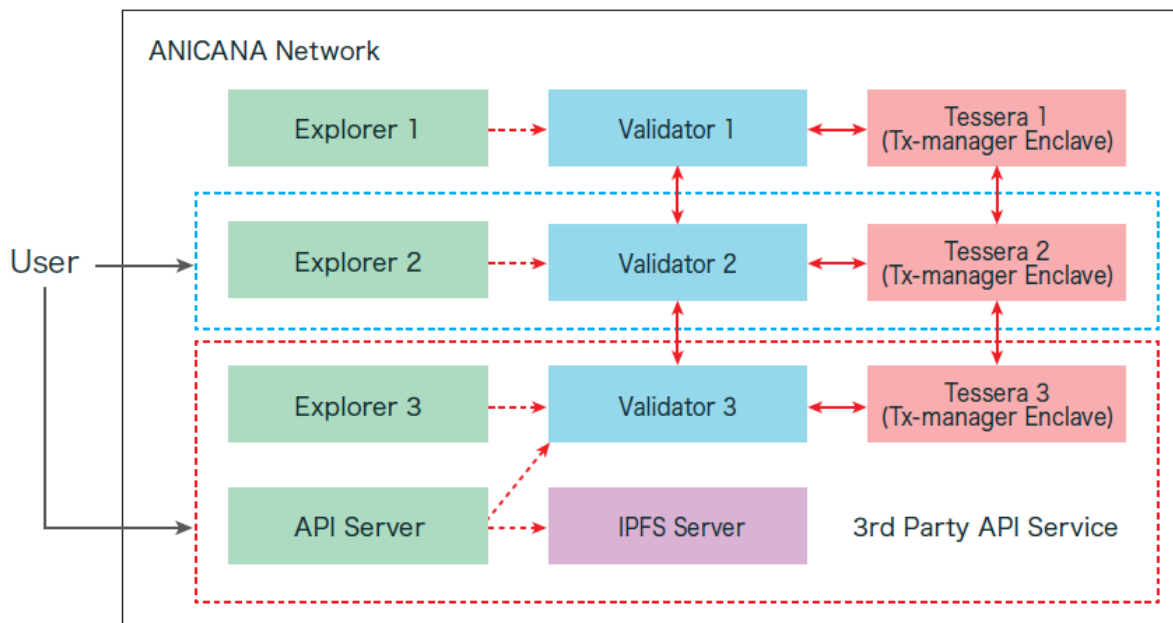


Fig. 1: ANICANA Consortium Chain

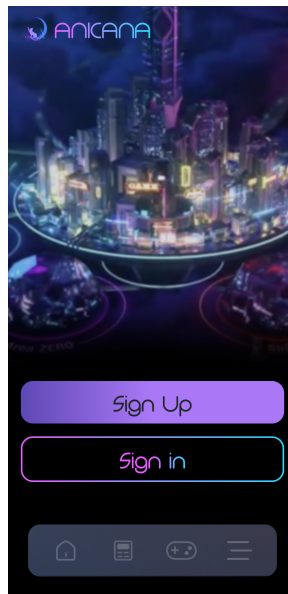
ANICANA WALLET REGISTRATION

The ANICANA portal site provides a wallet user interface, allowing you to generate a wallet by authenticating with your email address.

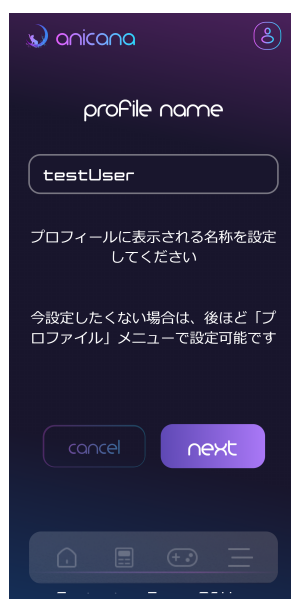
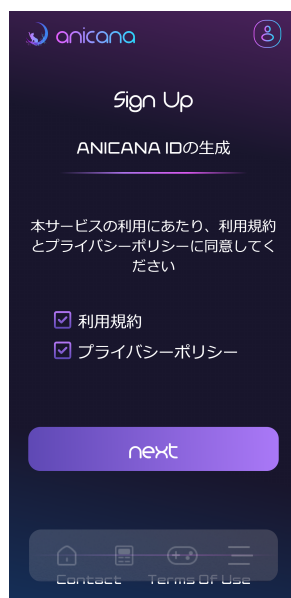
Please refer to the site URL for environment-specific information.

28.1 Wallet Registration Procedure

1. Click “sign up” on the ANICANA portal site to register as a user.



2. Agree to the terms of use and privacy policy.
3. Register a profile name.
4. Upload a profile picture.
5. Register your email address.
6. Enter the authentication code sent to your registered email address.
7. Register your phone number.
8. Enter the authentication code sent to your registered phone number.





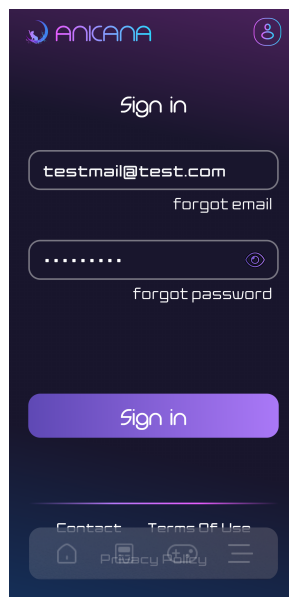
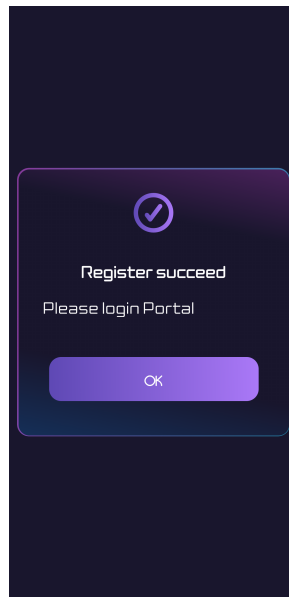


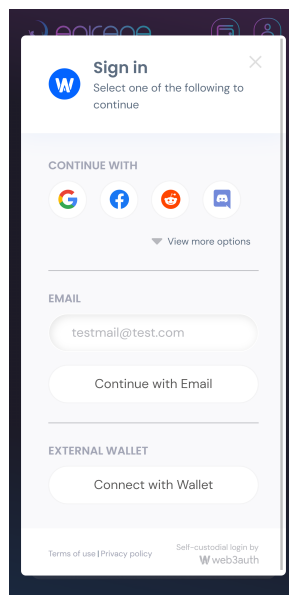
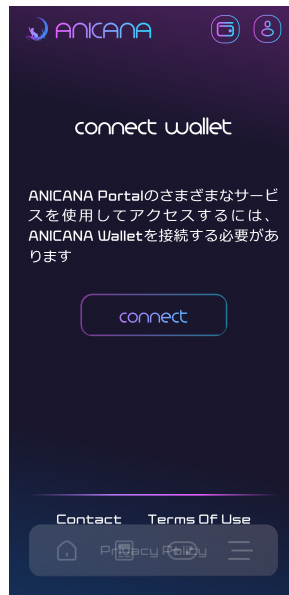


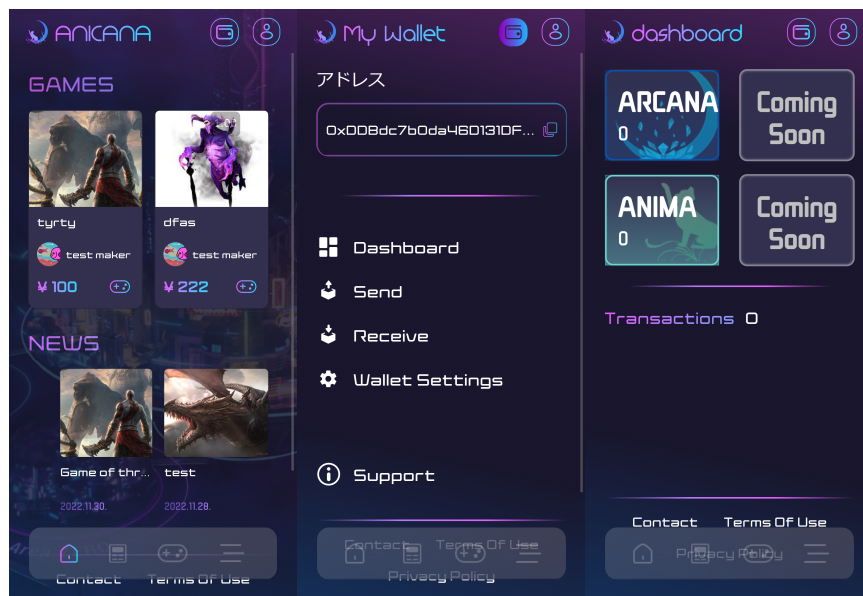
9. Set a password.



10. User registration is complete, and the registration completion modal will be displayed.
11. Sign in with the registered information.
12. On the wallet connection screen, click “connect.”
13. Enter your email address and click “Continue with Email.”
14. Wallet registration is complete. You can check your wallet address and token balance on the My Wallet page. Access the My Wallet page using the icon on the top right side.







PREPARING YOUR AWS ACCOUNT

Validator nodes are designed to be set up on Amazon Web Services (AWS). If you don't already have an AWS account, you can create one on the following website:

[Amazon Web Services](<https://aws.amazon.com>)

VALIDATOR SETUP

Set up the Validator on Amazon Web Service.

- Set up the ANICANA node.
- Set up the Validator Management UI.
- Configure the secret key for the Validator.

※ This step is performed by the ANICANA technical support team.

Here is the translation:

APPLY AS A VALIDATOR CANDIDATE

Apply to participate in the ANICANA network as a Validator. To apply, a specific amount of ANM tokens is required. You must send an application transaction to the smart contract while having the required amount of ANM tokens in your Validator account.

※ This step is performed by the ANICANA technical support team.

GENERATING SQUARE KEYS

Once your Validator application is accepted, you will need to generate Square Keys. You can generate up to five Square Keys per Validator.

Square Keys are key tokens required to generate EGG, which is essential for creating ARCANA NFTs. These keys can also be loaned to others.

Please note that this process will be carried out by the ANICANA Technical Support Team.

CONTENT DEVELOPMENT OVERVIEW

By connecting your content to ANICANA, you can offer your content's users the opportunity to generate ARCANA NFTs. Depending on the results of their interactions with your content, users can acquire ARCANA NFTs with their preferred images and names in their wallets.

ARCANA NFTs are generated from EGGs held by the publisher. Each EGG carries genetic information that influences the "parameters" of the generated ARCANA NFTs. Additionally, when generating ARCANA from EGGs, content providers must provide a seed value to the ARCANA NFTs, which also affects the "parameters."

33.1 Implementation Flow

The necessary steps for preparation are as follows:

Step	Details
Validator Setup	Refer to <i>Validator Setup</i> .
Generate Square Key	Refer to <i>Generate Square Key</i> .
Matrix Development Request	Refer to <i>Request Matrix Development</i> .
Acquire ANIMA	Refer to <i>Acquire ANM (ANIMA)</i> .
Generate EGGs	Refer to <i>Generate EGGs</i> .

Here are the tasks that can be performed in content development:

Step	Details
Connect Content and Wallet	Refer to <i>Connect Wallet</i> .
ARCANA Generation API Integration	Refer to <i>ARCANA Generation Flow</i> .
LEVICA Payment	Refer to <i>LEVICA Payment</i> .
Introduction of PERSONA	Refer to <i>PERSONA</i> .

ARK.ONE ENVIRONMENT INFORMATION

Ark.one Environment List

Caution:

Ark.one is a community-provided testnet.
Due to its testing nature, transactions issued within the testnet are not guaranteed.
Please refrain from actions involving real money within the testnet.

34.1 Environment Information

Item	Description
Chain ID	222221
JSON-RPC	https://stgchains.anicana.org/

34.2 Contract Addresses

Contract Name	Address
Anima	0x18F4a9E35d99E8E736f31eF11aA36F5D4ce7023c
Arcana	0xd7639Fc0cD23984b7F1F250803F0a7ad9D1eFAa8
Decomposer	0x837B0315d7dCB5ffaAc91b2dC085528c1fF75CE0
Egg	0x266Dc32CeabC06bC54469E8FAd9bE65efAfb66E8
EggBuilder	0xEcA31401263042B2Ec98457D0Ae1CaB9064f948E
Incubator	0x9C4EE916C997A469802A3F91ff729350A708C1cF
MatrixMaster	0x39BaC9943e4266096854029141867592E7958D3F
Shard	0x4Ca7323b9fB0EEc64ff23De4dCC67f434626FcEd
Square	0x8D3c73943b5ec3b64aeA43CD197F4214b0E70C38
ArcanaGeneratorInfo	0xCa59B3373F247F115D5A867CB0E2b18cAA43C96d
EggSupplement	0xB93181E64ea17B24a1a13dC31396CA86CE63B2c8
SquareSupplement	0x45BbC4fABbA1883A94BE0ff4Ab02d19B778d86bd
ContentsScopeApprover	0x9617Ba1f8a08B75b3d9E6fF4dE3E1233440969AB
AbsorbAuthority	0x36fddE6a2cCF18553B18e8C6b1A5535ee3B9cED1
AbsorbIntervalApprover	0x2D6296a287e4211c4b6c8232Baf0e3E084f6db7F
DrawChain	0x4B4DB59Fc8612D697D418CF9A6C39634E08B6589
Persona	0xD513eAd11bAfeE6b2bC08436a512FF3A8AC0265E
DrawAbilityLimiter	0x37Ef8Fa8995c43C26AC675C3A9DF317f3ed3A476
DrawPersonaCategoryLimiter	0xb9AA5Df1637d0cB103BA2F2D799AE878F082BBDF
DrawQuantityLimiter	0xF8BD4c1EEd08c48D086EB3bDeAE9eF03B05488aF
DrawFollowerLimiter	0xE4FDF41364930Fc4E43332E35d0F86478a0fa7D3
DrawCountLimiter	0xD1128FF78c7ba79B3f8E679387D6C0C53321482b
DrawPersonaLimiter	0xFcd33F400399f7E46dca668A4Ec36652e035c276
Boloodline	0xF062Fa680057396a9c6a336139E531E0FFaBfAd6

34.3 Contract ABI

Contract	ABI
Egg	Egg.json
ArcanaGeneratorInfo	ArcanaGeneratorInfo.json
EggSupplement	EggSupplement.json
SquareSupplement	SquareSupplement.json
ContentsScopeApprover	ContentsScopeApprover.json
AbsorbAuthority	AbsorbAuthority.json
DrawChain	DrawChain.json
Persona	Persona.json
DrawAbilityLimiter	DrawAbilityLimiter.json
DrawPersonaCategoryLimiter	DrawPersonaCategoryLimiter.json
DrawQuantityLimiter	DrawQuantityLimiter.json
DrawFollowerLimiter	DrawFollowerLimiter.json
DrawCountLimiter	DrawCountLimiter.json
DrawPersonaLimiter	DrawPersonaLimiter.json
Square	Square.json
Boloodline	Bloodline.json

34.4 Interfaces

Interface	Download
IDrawChainAuthorizer	IDrawChainAuthorizer.sol
IAbsorbApprover	IAbsorbApprover.sol

34.5 Libraries

Library	File
genSig	genSig.js
genSig.cfg.json	genSig.cfg.json

Caution: Please set the chainId of the environment you are using in genSig.cfg.json. genSig.cfg.json is referenced by genSig.js. Please place them in the same folder.

34.6 ANICANA Portal Site

- ANICANA Portal Site (Test Environment)
-

34.7 Call ARCANA Generation Page Script

Environment	API Endpoint (base_url)
Testnet	https://staging.anicana.org/

34.8 Check Status

Environment	API Endpoint
Testnet	https://api-staging.anicana.org/

34.9 Login Script

Environment	API Endpoint (base_url)
Testnet	https://staging.anicana.org/

34.10 LEVICA

Environment	API Endpoint (base_url),URL
Staging	http://levica-stg-apialb-1782828167.ap-northeast-1.elb.amazonaws.com
Merchant Management Screen	http://stg.store.levica.io/login

34.11 IPFS

Item	Description
API Server Endpoint	https://stg.anicana-api.akqjt.io/
Swagger UI	https://stg.anicana-api.akqjt.io/docs#/
IPFS gateway	https://stg.anicana-api.akqjt.io/ipfs/

PRODUCTION ENVIRONMENT INFORMATION

List of Production Environments

35.1 Environment Information

Item	Description
Chain ID	222222
JSON-RPC	https://chains.anicana.org/

35.2 Contract Addresses

Contract Name	Address
Anima	0x8C391195dC7Cb68D125EF35b73c859037603E548
Arcana	0x3923fCc1a12F385165bBF722c50A22B1d18335CD
Composer	0x73FD0678B2cD71be54EFBE069489a498Fdf820D2
Egg	0x9c382dd80F9D0865a7fC0953BaB6EdDb186FBaBa
EggBuilder	0x4cA9451003aD629e47d0C47d1164d6B66693811d
Incubator	0xb8726FE32cE1E0163b478b1ceeb0170CCFeA0794
MatrixMaster	0xAfC75DD63b30c55a3610ffa447c98c8c88CA1d0c
Shard	0xD1cF6C92DE56C791e036fA4d21914213c6CBaC8a
Square	0x52AB107d2c3Fb91aE2028d72105Aa8Bb5C55E667
ArcanaGeneratorInfo	0x338A498Ac956B67730c667efD02252bE1E2615b7
EggSupplement	0x4ed738d18e91baE47479b98302fA5936872C676e
SquareSupplement	0xa576401d922Ec79c7B6b93637f1BAd1A72D20CfF
ContentsScopeApprover	0x1000c42284Fa3BFD2F37280Fe1b61ab56bC894AA
AbsorbAuthority	0x4033696a12Ce2f6fF7ADDfD1D9F80C31Ea55C12A
AbsorbIntervalApprover	0x677c1cF5b1A41A0dCeb954beAA30F18228B2c521
DrawChain	0x894aB05BF700BA567530EBB1C2e8E5319DdB4233
Persona	0xCA6f428D07b00837C047bbeBe4a75F993C2288c1
DrawAbilityLimiter	0x13DaD62abfa9AA67f2AcFcDC48007c16205D36D6
DrawPersonaCategoryLimiter	0x1373c316d6DFC796F8C85b29160694AA8D229291
DrawQuantityLimiter	0xFC645cfC6cF992726D82183FF9E19B1c7E811f10
DrawFollowerLimiter	0x4c5443f1A3A774c4ef2F1ea5915510ff3D17BC2f
DrawCountLimiter	0x6d8Eb682dF8AC49A418a92Cba35b053e24735237
DrawPersonaLimiter	0xd05634254e70b4290368901831D753D6fB4f3eca
Boloodline	0x59436fc484aC84c9301e993F12B128df27B19276

35.3 Contract ABI

Contract	ABI
Egg	Egg.json
ArcanaGeneratorInfo	ArcanaGeneratorInfo.json
EggSupplement	EggSupplement.json
SquareSupplement	SquareSupplement.json
ContentsScopeApprover	ContentsScopeApprover.json
AbsorbAuthority	AbsorbAuthority.json
DrawChain	DrawChain.json
Persona	Persona.json
DrawAbilityLimiter	DrawAbilityLimiter.json
DrawPersonaCategoryLimiter	DrawPersonaCategoryLimiter.json
DrawQuantityLimiter	DrawQuantityLimiter.json
DrawFollowerLimiter	DrawFollowerLimiter.json
DrawCountLimiter	DrawCountLimiter.json
DrawPersonaLimiter	DrawPersonaLimiter.json
Square	Square.json
Boloodline	Bloodline.json

35.4 Interfaces

IF	Download
IDrawChainAuthorizer	IDrawChainAuthorizer.sol
IAbsorbApprover	IAbsorbApprover.sol

35.5 Libraries

Library	File
genSig	genSig.js
genSig.cfg.json	genSig.cfg.json

Caution: Please set the chainId of the environment you are using in genSig.cfg.json. Also, genSig.cfg.json is referenced by genSig.js, so place it in the same folder.

35.6 ANICANA Portal Site

- ANICANA Portal Site (Production Environment)
-

35.7 ARCANA Generation Page Invocation Script

-table::

header-rows
1

align
center

“Environment”, “API Endpoint (base_url)” “Production”,[”https://anicana.org/”](https://anicana.org/)

35.8 check status

Environment	API Endpoint
Production	https://api.anicana.org/

35.9 Login Script

Environment	API Endpoint (base_url)
Production	https://anicana.org/

35.10 LEVICA

Environment	API Endpoint (base_url)
Production	http://levica-prod-apilb-1703316262.ap-northeast-1.elb.amazonaws.com

35.11 IPFS

Item	Description
API Server Endpoint	https://chainapi.octillion.jp/
Swagger UI	https://chainapi.octillion.jp/docs#/
IPFS gateway	https://ipfs.octillion.jp/

USER WALLET RETRIEVAL

While it is not necessary for content-side systems to directly interact with the blockchain, there may be a need to obtain user wallet information.

Therefore, we provide a method for associating user information held on the content side with wallet address information by performing owner authentication of the address through ANICANA wallet integration.

36.1 User Registration Flow

The following user registration flow is expected:

1. Users register an account with the content.
2. During account registration, user's wallet address will be retrieved when connecting the wallet. If the user does not have a wallet, one will be automatically generated during the initial connection.
3. Link the content account with the user's address and maintain the information on the content side.
4. Authentication of the address holder can be achieved through wallet integration, enabling login to the content through address authentication instead of methods like password authentication. This allows for a Single Sign-On (SSO) that can be shared with other content providers.

WALLET CONNECTION

37.1 API Specification

37.1.1 Login Script

Please refer to the respective environment information pages for environment details.

Sample Generation Script:

```
<script src="https://staging.anicana.org/login.js" id="anikana_login_script" data-call-  
id="99999999" data-sign-text="HELLO" data-callback="https://staging.anicana.org/test_  
login.html" data-logout="true" ></script>  
<div style='text-align: center'><button class='' onclick='__open_portal_login()'>Login</  
button></div>
```

- Omit unnecessary optional parameters along with the key of the parameter.

Pa- rame- ter	re- quired/o	Type	Description
id	re- quired	String	anikana_login_script (Do not change)
src	re- quired	URL	{endpoint}/login.js (Refer to the endpoint in the environment information page)
data- call-id	re- quired	Num- ber	A unique number for each publisher. This is used on the content side to determine where the user is returning from, among other functions. If this information is not specifically needed, 99999999 can be used.
data- sign- text	op- tional	String	Text to be signed (one-time token)
data- callback	re- quired	URL	Callback URL. After logging in, callId, sign, and address (user's wallet address) will be added as GET parameters and redirected.
data- logout	op- tional	Bool	If true, it forces a re-login. If false, it automatically logs in if there is session information, and forces a re-login if there isn't. If not specified, it is treated as false.
data- referral- code	op- tional	String	Set the referral code passed from the affiliate.Fixed at 64 alphanumeric characters.

- Sample of direct URL generation

Direct URL generation:

```
{endpoint}/login/idms/{:call-id}/{sign-text}?r={callback}&logout={logout}
```

Direct URL generation (with referral-code):

```
{endpoint}/login/idms/{:call-id}/{sign-text}?r={callback}&logout={logout}&referral_  
↪code=XXXXX
```

- data-sign-text

This can be configured for advanced security implementation.

For details, refer to [here](#).

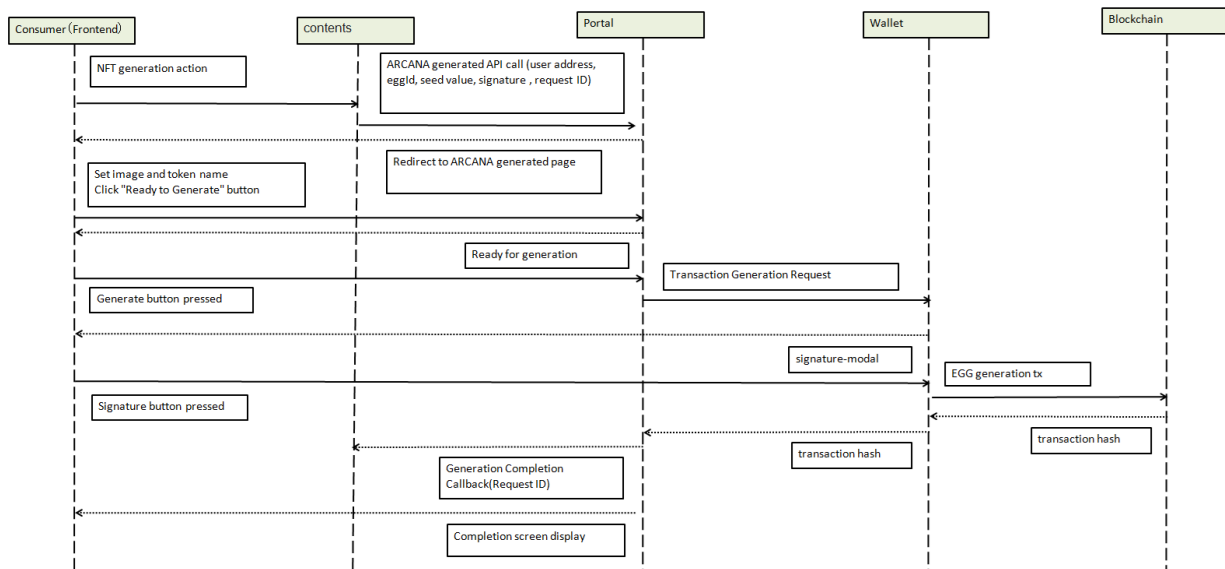
ARCANA GENERATION PROCESS

ARCANA generation is typically performed through the integration of IPFS and the ANICANA Chain smart contract.

- Upload token image to IPFS
- Generate and upload metadata JSON to IPFS
- Generate and possess EGGs
- Generate a signature to grant permission to open the possessed EGGs to users
- Transaction signing by the user
- Execution of the ARCANA generation transaction

Implementing these processes independently on the content side can be complex, so the ANICANA portal provides API support for these operations. Content providers can embed a script tag in their frontend and provide the necessary parameters to navigate users to the ARCANA generation page and facilitate ARCANA generation.

38.1 Integration Flow with ARCANA Generation Page (API)



38.2 ARCANA Generation on the Ark.one

ARCANA generation on the Ark.one can be performed using the following steps:

1. Register on the Validator management page with an email address and create a wallet.
2. Obtain the private key of the wallet address generated in the previous step. Use this private key to create a signature. The privatekey can be checked from the console of the browser's development tools by logging in to the Validator management page as the target user.
3. In the Ark.one environment, EGGs are not generated from the Validator management page but are issued by the technical support team with administrative privileges.

ARCANA GENERATION API

You can invoke the ARCANA generation screen by embedding the following script tag in your content.

39.1 API Specifications

39.1.1 Script to Invoke ARCANA Generation Page

Sample Generation Script:

```
<script src="https://staging.anicana.org/arcana.js" id="gen_arcana_script" data-  
  ↳requestid="9999999" data-toaddr="0xFf5BC900110f5c4eb6Ce2faf2081B4151655B3f3" data-seed=  
  ↳"10000" data-eggid="10" data-signature=  
  ↳"0xdfe893d3906b31c0cfcc05b05387c7cf3bf31524caeac2fb5e3d7b9d144dbc9550a9ce41d92ad4c070c6f34c38ba8329d8  
  ↳" data-callback="https://staging.anicana.org/test_button.html" data-logout="true" ></  
  ↳script>  
<div style='text-align: center'><button onclick="__go_to_arcana_generator()">Generate_  
  ↳ARCANA</button></div>
```

Parameter	required	Type	Description
id	required	String	gen_arcana_script (Do not change)
src	required	URL	{endpoint}/arcana.js (Refer to the environment information page for the endpoint)
data-eggid	required	Number	The EGG eggid held by the publisher.
data-seed	required	Number	Seed
data-signature	required	String	Publisher's signature. Refer to the signature generation procedure page.
callback-url	optional	URL	Callback URL. The requestId and txHash will be added as GET parameters and redirected. If you specify http://test.com , it will become http://test.com?requestId=1&txHash=xxxxx . You can also omit the callback, in which case a button will be displayed to navigate to the wallet page on the portal.
data-request	required	Number	Any number specific to the publisher (0 ~ 18446744073709551615). Used in check status.
data-toaddr	required	address	Wallet address for distributing ARCANA
data-logout	optional	boolean	If true, forcefully trigger a re-login. If false, automatically log in if there is a session, otherwise, prompt for re-login. If not specified, it is the same as false.
data-symbol	optional	String	Symbol that can be set by the publisher.
data-manaInfo	optional	String	Text that can be set by the publisher. It is envisaged to add value to ARCANA by writing things like user experience information or encrypted personal information in content.
data-manaValue	optional	Number	Numeric value that can be set by the publisher.
data-manaAddress	optional	address	Specify the manaAddress of the interrupted ARCANA generation.

To directly call the generation page, do as follows:

```
{endpoint}/arcana-gen/{eggId}/{seed}/{signature}/{requestId}/{toAddress}?r={callbackUrl}
↪&logout=true

(with mana information)
{endpoint}/arcana-gen/{eggId}/{seed}/{signature}/{requestId}/{toAddress}/{symbol}/
↪{manaInfo}/{manaValue}?r={callbackUrl}&logout=true

(with manaAddress specified)
{endpoint}/arcana-gen/{manaAddress}
```

supplement

- When calling the generation page directly and not specifying symbol, manaInfo, and manaValue, please insert null in the respective locations.
- The maximum number of “manaInfo” characters is limited to the total of all request headers. Although there are some conflicts with other parameters, the maximum number of characters for symbol, manaInfo, and manaValue together should be 800 or less for Japanese and 7200 or less for single-byte alphanumeric characters.

- The text to be displayed in the mana information is currently not line breakable.

39.1.2 Check Status

Retrieve the status of ARCANA generation.

Method:

GET

Endpoint:

/api/arcana-status/{wallet_address}/{request_id}

Parameter	Description
wallet_address	Address of the signer (address of the EGG holder)
request_id	Request ID specified when calling the ARCANA generation API

Sample response

```
{
  "data": {
    "status": "done",
    "transaction_id":
    ↪ "0x2e35551b1bf7bb6942610be99dcf60fafe804f167c19a2070c45ff1a0a7f50de"
  },
  "status": "success"
}
```

Value of status (inside data)

Status	Description
no_transaction	User has not yet completed the ARCANA generation process. (Including cases where the user exited)
transaction_created	ARCANA generation transaction has been sent to the blockchain but the result is not confirmed yet.
error	Transaction failed for some reason and terminated (ARCANA has not been generated).
done	ARCANA has been generated and the process completed successfully.

Error response

```
{
  "message": "request_id"
}
```

Note:

In case of error, a 404 status will be returned.

39.1.3 Flow to ARCANA Generation

The process for ARCANA generation follows a flow similar to the following:

1. Validator Setup.
2. Granting SHARD, ANIMA
3. Registering Matrix, Activating Matrix
4. Generating EGG in Validator Management Interface.
5. Obtaining the private key of the Validator from a dedicated site.
6. Creating a signature using the obtained private key.
7. Generating ARCANA using the EGG and signature created above.

In the staging environment, you can perform the following steps:

1. Register with an email address in the Validator Management Interface. A wallet will be created.
2. The privatekey is obtained by using the privatekey of the walletaddress issued above. privatekey can be checked from the console of the development tools in the browser by logging in to the Validator UI as the target user. Use the private key displayed with “0x” added at the beginning for creating the signature.
3. EGGs in the staging environment are issued by administrative authority, not by generating them from the Validator UI.
4. Set the issued EGG’s ID in the eggid parameter. You can check the EGGs you own in the Validator Management Interface.

GET A LIST OF OWNED EGGS

Sample to retrieve a list of EGGs currently owned by the user.

Please refer to the contract, JSON-RPC, and ABI files for test environment information.

Caution: Please use web3 version 1.9.8.

Example of Retrieving a List of EGGs (JavaScript):

```
var Web3 = require('web3');
var eggAbi = require("./egg.json");

const web3 = new Web3("https://stgchains.anicana.org/"); // Specify the JSON-RPC URL

const eggAddr = "0xb374640Ca3E3DA6F836ca8c60130fCAE2da3B929"; // Specify the address of
↳ the Egg contract
const holderAddr = "0xe092b1fa25DF5786D151246E492Eed3d15EA4dAA"; // Address for which
↳ you want to check the EGG ownership

const eggContract = new web3.eth.Contract(eggAbi, eggAddr);

const listOfEggs = async () => {

    var balance = await eggContract.methods.balanceOf(holderAddr).call();
    console.log(balance);

    var eggIds = [];
    for (var i = 0; i < balance; i++) {
        var res = await eggContract.methods.tokenOfOwnerByIndex(holderAddr, i).call();
        eggIds.push(res);
    }

    console.log(eggIds);
}
```

Batch Retrieval of Owned EGGs Function:

```
@param account Address of the account holding the tokens
@param index Index number of the token to retrieve
@param limit Maximum number of tokens to retrieve
```

(continues on next page)

(continued from previous page)

```
@return Array of token information
// You can retrieve up to around 1000 tokens in one call. The limit depends on the state
↳ of the smart contract, but if you exceed the limit, an error will be returned.
function tokenOfOwnerByIndexBatch(address owner, uint256 index, uint256 limit) public
↳ view returns (uint256[] memory)
```


SIGNATURE GENERATION PROCEDURE

In order to generate an ARCANA token, the Validator (on the content side) needs to allow the user to convert their EGG token into ARCANA. Additionally, during this process, it is necessary to embed the result value of the content into the ARCANA token without tampering with this value. To achieve this, the following procedure is performed: the content side creates a signature, and the user uses this signature to send a transaction. The private key of the eggid owner can be checked from the console of the development tools in the browser by logging in to the Validator UI as the target user.

41.1 Creation of Signature Data for ARCANA Generation

To create signature data, the following data is required.

Based on the above data, create the dataToBeSigned, which is the data to be signed, using the following steps:

```
const genSig = require("./genSig.js");  
  
const signature = genSig.signForIncubate(eggid, toAddr, seed, contract, privateKey);
```

41.2 Creation of Signature Data for PERSONA Distribution

Based on the above data, create the signature data using the following steps:

```
const genSig = require("./genSig.js");  
  
const sigInfoApp = genSig.signForPersonaApprove(to, tokenId, contract, privateKey);  
  
const sigInfoAppNonce = sigInfoApp.nonce;  
const sigInfoAppSign = sigInfoApp.sign;  
  
const sigInfoTrans = genSig.forPersonaTransferFrom(from, to, tokenId, contract,   
↪privateKey);  
  
const sigInfoTransNonce = sigInfoTrans.nonce;  
const sigInfoTransSign = sigInfoTrans.sign;
```

41.3 Libraries

Refer to the environmental information.

LEVICA PAYMENT

This is an electronic money application available for prepaid payments which is used for various products and integrated services deployed on the ANICANA network.

42.1 API URL Format

The URI for the LEVICA REST API follows the format below:

```
`${base_url}/v${version}/${resource}.
```

Field	Description
base_url	URL of the server hosting the API. 'base_url' is the same for all APIs.
version	API version.
resource	Unique name assigned to each API.

42.2 Environment Information

Please refer to each environment information page.

42.3 Request Authentication

The LEVICA system exchanges data in a RESTful format.

- Request Authentication

To access resources of the system using the REST API, merchant authentication is required first. Authentication is performed using the merchant's clientId and clientSecret. The system provides a login REST API to execute the authentication. The clientId and clientSecret are passed to the system with this request. The system compares whether the clientId and clientSecret match the values in the database and checks them. If they match, the system creates an access token specific to the merchant, and these tokens are returned to the caller as a response to the authentication request.

Access tokens contain the necessary merchant information, and these access tokens need to be stored. Therefore, when the system receives an access token with a REST API request, it can verify the merchant's authentication information.

Pattern 1

Header Key	Type	Required	Description
Content-Type	String	Yes	Content type of the request body.
Accept	String	No	Indicates the content type.

Pattern 2

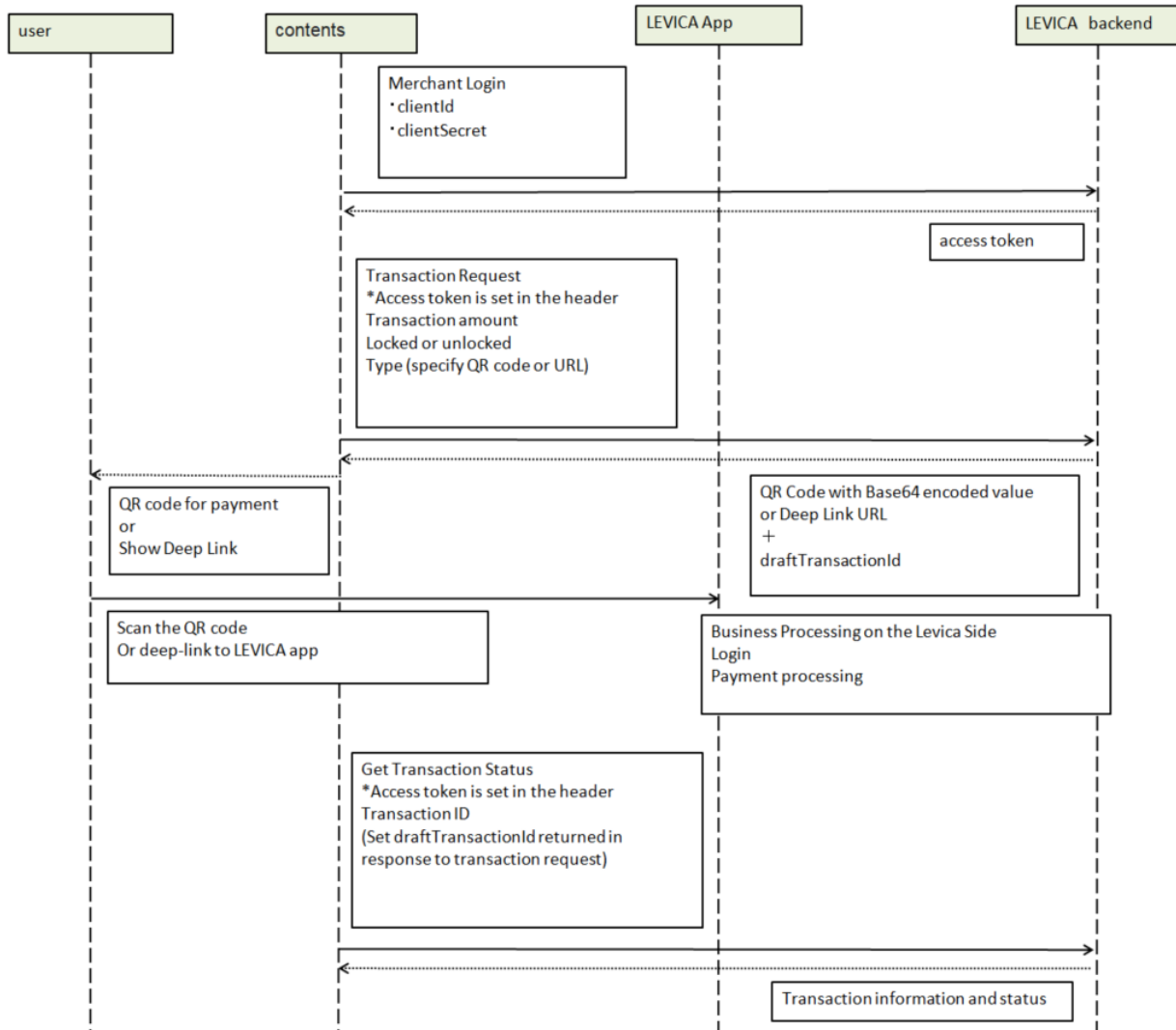
Header Key	Type	Required	Description
Authorization	String	Yes	Access token.
Content-Type	String	Yes	Content type of the request body.
Accept	String	No	Indicates the content type.

For browsers, the device type is not required.

If the API has an option for file upload, the Content-Type will be 'multipart/form-data.' For other POST/PUT APIs, the Content-Type will be 'application/json.'

Caution: You must first apply for merchant registration with LEVICA and obtain the clientId and clientSecret.
--

42.4 Integration Flow with LEVICA



42.5 Main APIs for Integration

Listed below are the main APIs used to implement LEVICA Payment.

- Merchant Login

- Transaction Request

Request URI	<code>\${base_url}/\${v{version}}/merchant/transaction</code>			
Method	post			
Objective	Transaction Initiation			
Request				
Request Header	Content-Type: application/json Accept: application/json Authorization: merchant_access_token			
Request Parameters	Field	Type	Required	Description.
	amount	Long	Yes	Amount of transaction
	isLock	byte	No	0 : No lock (default setting) 1 : lock
	type	byte	Yes	1 : Generate QR Code 2 : Generate deep link URLs
	referralCode	String	No	Set the referral code passed from the affiliate Fixed 64 alphanumeric characters
Sample request body	<pre>{ "amount": 10000, "isLock": 1, "type": 1, "type": 1, "referralCode": "123456789101234567891234567890ASDFGHJKLZXCVBN- MASDFGHJKASDFGHJKAS" }</pre>			
Response				
Success Response	Http status code: 200 type=1 <pre>{ "type": 1, "hasLock": 1, "data": "<Base64_Encoded_String>", "draftTransactionId": "D102656693ac3ca6e0cdafbfe89ab99", "value": "<Deep Link URI>", "createdDate": "2022-09-1T18:25" }</pre>			

- Get Transaction Status

Request URI	<code>\${base_url}/\${v{version}}/merchant/transaction/{transactionID}/status</code>			
Method	get			
Objective	Get transaction status			
Request				
Request Header	Content-Type: application/json Accept: application/json Authorization: merchant_access_token			
Request Parameters	Field	Type	Required	Description.
	transactionID	String	Yes	Draft-TransactionID obtained from TransactionRequestAPI
Sample request body	Empty			
Response				
Success Response	Http status code: 200 <pre>{ "tempTransactionID": "D5a321108871ea447db69a56404ad65ae46d0073bc68fa91fc60f579f8305ec4b", "transactionId": "4833ea425b55599d97dd700878e0c3a4bf5e276e70edb8636344aa434447bd56", "isLock": 1, "type": 1, "status": 3, // 1 => pending, 2 => Payment completed, 3=> Transaction completed successfully, 4=> transaction fail, 5=> transaction canceled. Additional status information is provided outside the column. "amount": "500", "fromAddress": "0x5J3mBbAH58CpQ3Y5RNJpUKP", "toAddress": "0xPKUpJNR5Y3QpC85HAbBm3J5", "transactionCreateDate": "2022-08-16T09:21:49.000+00:00", "transactionPaymentDate": "2022-08-16T10:21:49.000+00:00", "transactionCompleteDate": "2022-08-17T09:21:49.000+00:00" }</pre>			
Error Response	Http status code: 401, Unauthorized <pre>{ "message": "Invalid access token", "code": "6001" }</pre> Http status code: 404, Not Found <pre>{ "message": "No transaction found", "code": "1006" }</pre>			

※If status is 2 or more, the settlement can be considered complete; if status 4 or 5 we can conclude that there is no problem with the content side because the content is at the blockchain level.

42.6 Testing in the Staging Environment

In the staging environment, you can perform tests by charging the balance using test card numbers. The payment system uses Stripe, so you can use the following card numbers:

Card Company	Card Number			Expiration Date	Security Code			Other Fields	Form
Visa	4242	4242	4242	Valid future date	Any code	3-digit	security	Any value	
Visa (Debit)	4000	0566	5566	Valid future date	Any code	3-digit	security	Any value	
Mastercard	5555	5555	5555	Valid future date	Any code	3-digit	security	Any value	
Mastercard (Debit)	5200	8282	8282	Valid future date	Any code	3-digit	security	Any value	
Mastercard (Pre-paid)	5105	1051	0510	Valid future date	Any code	3-digit	security	Any value	
American Express	3782	822463	10005	Valid future date	Any code	4-digit	security	Any value	

PERSONA OVERVIEW

PERSONA is an NFT that plays the role of a catalyst to trigger specific smart contracts on ANICANA. The smart contracts are ERC721 compliant.

They are primarily created by publishers and offered to Consumer (content users).

The methods of offering may vary, including paid options and rewards in events.

The generation of PERSONA requires both SQUARE and ANIMA.

PERSONA has internal values, and Validators can set these internal values when generating PERSONA.

43.1 PERSONA Growth / Object Absorption Contract

PERSONA can execute smart contracts to absorb (Absorbing) ARCANA objects (tokens) up to a specified number of times.

Depending on the internal values of the objects being absorbed, the internal values of the absorbed PERSONA change, and the absorbed objects disappear.

This characteristic allows for the creation of various added values.

PERSONAs with high FORCE internal values primarily target ARCANAAs with low FORCE values.

When a low-FORCE PERSONAs absorb a high-FORCE ARCANAAs, there is a higher probability that the internal values of the absorbing PERSONAs will deteriorate.

Consumer can enjoy the changes resulting from object absorption, and they can also utilize PERSONAs for collecting or selling high-ability PERSONAs on secondary marketplaces.

43.2 DrawChain / Contract to Retrieve Specific Data

PERSONAs come with smart contracts called DrawChain, and when DrawChain is activated for SQUAREs, Validators can retrieve specified data (services) with the designated PERSONA.

The internal values of PERSONA can be set as conditional criteria to draw (retrieve) the data (services).

The usage history of DrawChain is recorded on the blockchain, and it can be limited, such as one-time use in the same event.

By developing PERSONA to increase its abilities (internal values), players can engage in secondary trading. Publishers can distribute limited items within their content events using DrawChain on SQUARE. This enables a variety of productions.

Publishers can utilize DrawChain to distribute various items such as NFTs, tickets, cryptocurrencies, and points, creating attractive events.

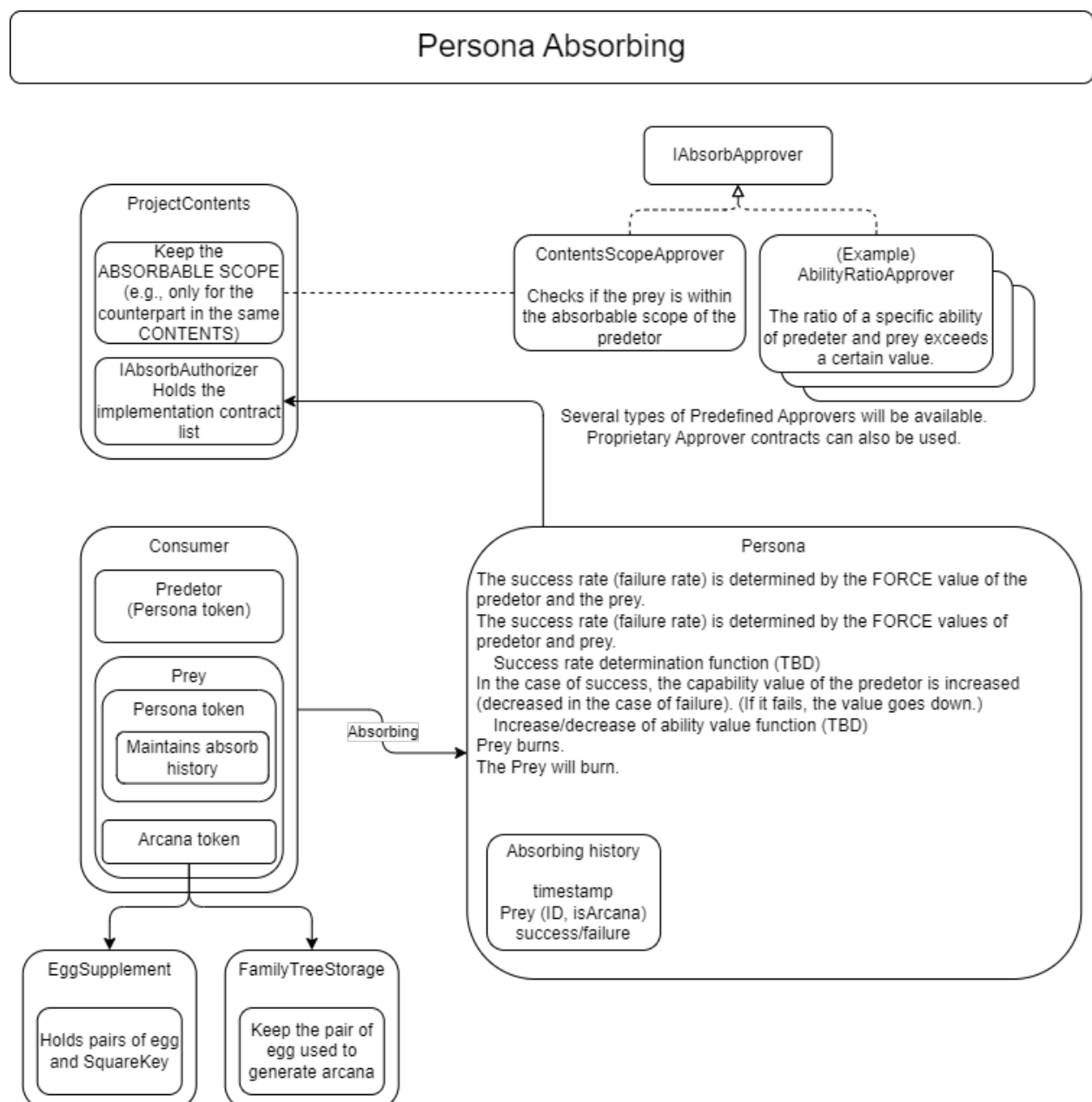
PERSONA IMPLEMENTATION PROCEDURE

The implementation of PERSONA follows the following steps.

Step	Details
Setting up Absorb	Refer to <i>Setting up Absorb</i> .
Setting up DrawChain	Refer to <i>Setting up DrawChain</i> .
Generating and Distributing PERSONA	Refer to <i>Minting PERSONA</i> .
User Utilization of PERSONA	Refer to <i>User Utilization of PERSONA</i> .

ABSORBING

45.1 Overview Diagram



45.2 Setting Absorbing Conditions

Publishers can set the absorption targets for PERSONA. (Unlimited targets are also possible)

45.2.1 Prerequisites

Reverting occurs if the prerequisites are not met.

The predator (PERSONA) must own the address calling the function.

It must be a combination of predator (PERSONA) and prey (ARCANA) permitted by AbsorbAuthority.

45.2.2 Restricting the Execution of absorb()

As mentioned in the prerequisites, absorb() must be a combination of predator and prey permitted by AbsorbAuthority.

To restrict the execution of absorb(), you need to configure AbsorbAuthority.

AbsorbAuthority can operate contracts that inherit multiple IAbsorbApprover interfaces for each PERSONA category of Predator.

Except for numAbsorbApprovers, only the owner of the square key can operate the following functions.

The steps are as follows.

Create a contract to restrict the execution of absorb()

Create a contract that implements the IAbsorbApprover interface (IAbsorbApprover.sol):

```
@param presetId Preset number (freely defined in the implementation contract)
@param predator PERSONA Id on the absorbing side
@param prey PERSONA Id on the absorbed side
@return true: absorbable, false: not absorbable
function approveAbsorb(uint256 presetId,uint256 predator,uint256 prey) external view
↳ returns (bool)
```

Refer to the environment information for the interface file.

Register the created contract

Register the contract using the function (AbsorbAuthority.sol):

```
@param contentsId Content ID (contentsId & SquareKey)
@param contractAddr Contract address that inherits IAbsorbApprover
@param presetId The presetId of contractAddr, the usage of preset Id depends on the
↳ implementation contract of IAbsorbApprover.
@return Index of the AbsorbApprover array for each registered Content ID
function addAbsorbApprover(uint32 contentsId,address contractAddr,uint256 presetId)
↳ public returns (uint256)
```

45.2.3 Other Functions for Setting absorb() Conditions

Delete registered ApproverInfo (AbsorbAuthority.sol)

```
@param contentsId Content ID (contentsId & SquareKey)
@param idx Index of the AbsorbApprover array for each Content ID
function removeApprover(uint32 contentsId,uint256 idx)
```

Replace registered ApproverInfo (AbsorbAuthority.sol)

```
@param contentsId Content ID (contentsId & SquareKey)
@param idx Index of the AbsorbApprover array for each Content ID
@param contractAddr Contract address that inherits IAbsorbApprover
@param presetId The presetId of contractAddr, the usage of preset Id depends on the
↳ implementation contract of IAbsorbApprover.
function replaceApprover(uint32 contentsId,uint256 idx,address contractAddr,uint256
↳ presetId)
```

Get the number of registered AbsorbApprovers for each Content ID (AbsorbAuthority.sol)

```
@param contentsId Content ID (contentsId & SquareKey)
@return Number of registered AbsorbApprovers for each Content ID
function numAbsorbApprovers(uint32 contentsId) public view returns (uint256)
```

AbsorbApprover list for each Content (AbsorbAuthority.sol)

```
@notice Available with approverList(contentsId)
mapping(uint32 => ApproverInfo[]) public approverList;
```

ApproverInfo

```
struct ApproverInfo {
    @notice Address of the AbsorbApprover contract
    address approver;
    @notice Preset ID of the AbsorbApprover contract to be used
    uint256 presetId;
}
```

Get SquareKey from arcanald (EggSupplement.sol)

```
@param arcanaId ARCANA token ID
@return SquareKey associated with arcanaId
function arcanaToSquareKey(uint256 arcanaId) external view returns (uint256)
```

45.2.4 Implemented IAbsorbApprover

Currently, the following contracts implementing the IAbsorbApprover interface are available.

To enable them, you need to register the contracts with AbsorbAuthority using addAbsorbApprover().

Contracts Limited by Square Key

(ContentsScopeApprover.sol)

A contract that limits absorbability based on the square key associated with PERSONA and ARCANA

PERSONA of the predatorContents specified in the function can only target ARCANAs of the preyContents specified category.

To enable absorbing restrictions by ContentsScopeApprover, add restrictions with the following function and register the contract with AbsorbAuthority.

To add restrictions with ContentsScopeApprover, use the following function.:

```
@notice Set the absorbability list for each content
@param predatorContents Predator (PersonaId)
@param preyContents      Prey (ArcanaId)
@param mask               0xffff0000 (the lower 16 bits are unused)
@param arc               Always set to true and 0
function setAbsorbScope(uint32 predatorContents,uint32 preyContents,uint32 mask,uint8
↳arc) public;
```

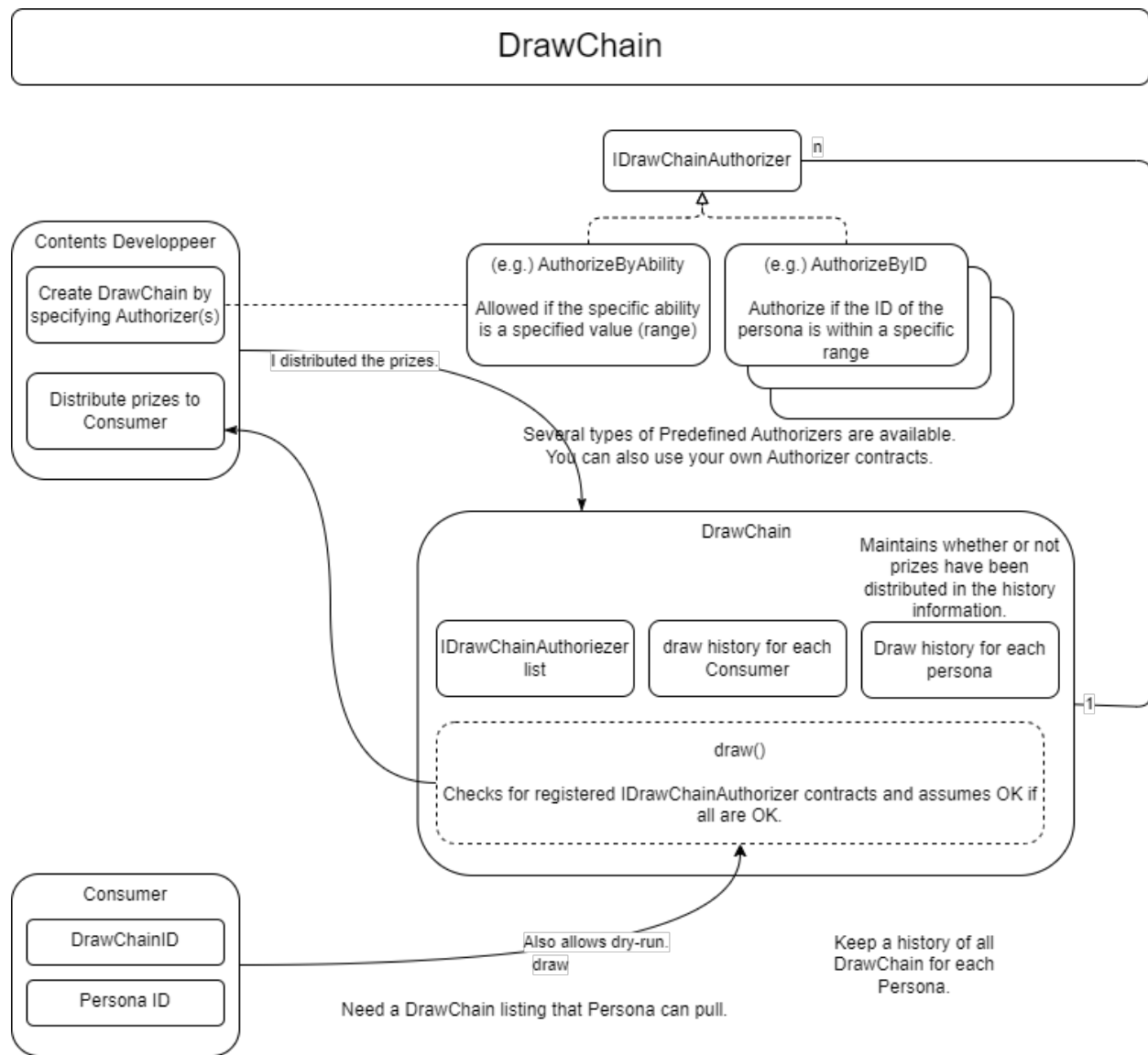
Contract for Time Restriction on Re-execution

(AbsorbIntervalApprover.sol)

A contract that prevents the execution of absorbing until a certain period has elapsed since the previous absorbing. Set the time in milliseconds for re-absorbability when registering the contract with AbsorbAuthority using the presetId.

IMPLEMENTATION OF DRAWCHAIN

46.1 Overview



46.2 Setting Up DrawChain

You can configure the conditions for running DrawChain.

Configuration is possible by registering a contract that inherits the IDrawChainAuthorizer interface.

Functions for Publishers

1. Create a contract to set conditions for running DrawChain:

Create a contract that implements the IDrawChainAuthorizer interface (IDrawChainAuthorizer.sol):

```
@param drawChainId DrawChain ID
@param presetId Preset number (freely defined within the implementation contract)
@param personaId PERSONA Id
function authorizeDraw (uint256 drawChainId, uint256 presetId, uint256 personaId)
↳external view returns (bool);
```

Refer to the environment information for the interface file.

2. Register DrawChain

Register the contract created in step 1 in AuthorizerInfo. Set the address of the created contract and presetId in AuthorizerInfo. The usage of presetId can be freely defined within the implementation contract.

Function to register DrawChain (Drawchain.sol):

```
@param squareKey SquareKey associated with DrawChain
@param _authorizers List of authorizers
@return DrawChain ID
function newDrawChain(uint256 squareKey, AuthorizerInfo[] calldata _authorizers) public
↳returns (uint256)
```

AuthorizerInfo[]:

```
struct AuthorizerInfo {
    @notice Address of the IDrawChainAuthorizer contract
    address authorizer;
    @notice Preset ID of the IDrawChainAuthorizer contract
    uint256 presetId;
}
```

46.2.1 Setting Active and Inactive States

During the initial registration, draw capability is set to true.

DrawChain creators can control the execution of draw. They can change the state to active (true) or inactive (false).

Function to set the active and inactive states of DrawChain (Drawchain.sol):

```
@param drawChainId DrawChain ID @param active true: draw is possible, false: draw is not possible
function deactivateDrawChain(uint256 drawChainId, bool active) public;
```

46.3 Other DrawChain Functions

Returning an Array of DrawChain IDs for Which a Specific PERSONA Can Draw (Drawchain.sol):

```
@param from Starting ID of DrawChains to inspect (inclusive)
@param until Ending ID of DrawChains to inspect (inclusive)
@param limit Limit the search to this many successful DrawChains (maximum number of
elements to return in the array)
@param personaId PERSONA ID to draw DrawChains
@return Array of DrawChain IDs where draw is successful
function disponibles(uint256 from, uint256 until, uint256 limit, uint256 personaId)
public view returns (uint256[] memory)
```

Regarding Specified Ranges

Be cautious when using disponibles() to search over a wide range, as it may result in gas exhaustion.

Getting DrawChain Information (Drawchain.sol):

```
@param fromId Starting DrawChain ID
@param count Number of DrawChain information to retrieve
@return Array of DrawChainInfo
function getDrawChain(uint256 fromId, uint256 count) public view returns
(DrawChainInfo[] memory)
```

DrawChainInfo:

```
struct DrawChainInfo {
    uint256 id;
    uint32 squareKey;
    uint8 active;
    uint8 pad1;
    uint16 pad2;
    uint64 pad3;
    uint128 pad4;
}
```

Returning the Number of Draws (History Count) for Each DrawChain (Drawchain.sol):

```
@param drawChainId DrawChain ID
@return Number of draws (history count)
function drawHistoryCountByDrawChain(uint256 drawChainId) public view returns (uint256)
```

Returning the Draw History for Each DrawChain (Batch Version) (Drawchain.sol):

```
@param drawChainId DrawChain ID
@param fromIdx Starting index (inclusive)
@param count Number of draw histories to retrieve
@return Array of draw histories
function drawHistoryByDrawChain(uint256 drawChainId, uint256 fromIdx, uint256 count)
public view returns (History[] memory)
```

Returning the Number of Draws (History Count) for Each PERSONA (Drawchain.sol):

```
@param personaId PERSONA Id
@return Number of draws (history count)
function drawHistoryCountByPersona(uint256 personaId) public view returns (uint256)
```

Returning the Draw History for Each PERSONA (Batch Version) (Drawchain.sol):

```
@param personaId PERSONA Id
@param fromIdx Starting index (inclusive)
@param count Number of draw histories to retrieve
@return Array of draw histories
function drawHistoryByPersona(uint256 personaId, uint256 fromIdx, uint256 count) public
↳view returns (History[] memory)
```

History:

```
struct History {
    @notice History ID, same as the value returned by draw()
    uint256 id;
    @notice DrawChain Id
    uint256 drawChainId;
    @notice PERSONA Id
    uint256 personaId;
    @notice Owner of the PERSONA at the time of the draw
    address personaOwner;
    @notice Timestamp when the draw occurred
    uint128 drawnOn;
    @notice Timestamp when the delivery was made
    uint128 deliveredOn;
}
```

Returning the Number of Draws (History Count) for Each DrawChain and PERSONA (Drawchain.sol):

```
@param drawChainId DrawChain ID
@param personaId PERSONA Id
@return Number of draws (history count)
function drawHistoryCountByDrawChainAndPersona(uint256 drawChainId, uint256 personaId)
↳public view returns (uint256)
```

Returning the Draw History for Each DrawChain and PERSONA (Batch Version) (Drawchain.sol):

```
@param drawChainId DrawChain ID
@param personaId PERSONA Id
@param fromIdx Starting index (inclusive)
@param count Number of draw histories to retrieve
@return Array of draw histories
function drawHistoryByDrawChainAndPersona(uint256 drawChainId, uint256 personaId,
↳uint256 fromIdx, uint256 count) public view returns (History[] memory)
```

Returning the Number of Draws (History Count) for Each PERSONA Owner (Drawchain.sol):

```
@param personaOwner Persona owner address
@return Number of draws (history count)
function drawHistoryCountByPersonaOwner(address personaOwner) public view returns
↳(uint256)
```


Returning the Draw History for Each PERSONA Owner (Batch Version) (Drawchain.sol):

```
@param personaOwner Persona owner address
@param fromIdx Starting index (inclusive)
@param count Number of draw histories to retrieve
@return Array of draw histories
function drawHistoryByPersonaOwner(address personaOwner, uint256 fromIdx, uint256 count)
    public view returns (History[] memory)
```

46.4 Executing DrawChain

1. Drawing a DrawChain Contract: Drawchain

Function Executed by Users When Performing Operations

Function to draw a DrawChain (Drawchain.sol):

```
@param drawChainId DrawChain ID @param personaId PERSONA ID @return 0: Draw failed.
Non-zero: Index of the history function draw(uint256 drawChainId, uint256 personaId) public returns
(uint256)
```

2. Calling the DrawChain creator (Publisher) when distributing prizes and register the timestamp when delivery of prizes is made.

Function for Publishers

Function to register a timestamp (Drawchain.sol):

```
@param historyId History ID returned when draw is successful function delivered(uint256 historyId)
```

About Timestamps

The delivered() function is optional.

When called upon delivering prizes, a timestamp is registered in the deliveredOn field of the History structure.

If it is not executed, the only consequence is that the delivery history will not be stored at the blockchain level.

The advantages of performing this include:

- Timestamps are set at the blockchain level and cannot be tampered with
 - It can be used for future integration with other programs on the smart contract
-
-

46.5 Implemented IDrawChainAuthorizers

The following contracts currently implement the IDrawChainAuthorizer interface and are available for use. To enable them, you need to set the contract in the AuthorizerInfo during DrawChain registration.

46.6 Contract to Limit the Ability Values of PERSONAs that Can Draw (DrawAbilityLimiter.sol)

(DrawAbilityLimiter.sol)

The ability values need to be set by the square key owner in advance.

After setting the values, set the contract in AuthorizerInfo during DrawChain registration.

If the ability values of the PERSONA to be drawn are within the set range, drawing becomes possible.

Functions for Registration:

```
@param limit Set the ability value limits. Limit[6] corresponds to FOR, ABS, DFT, MND,
↳INT, EXP, in that order.
@return numPresets Registration number
function newPreset(Limit[6] calldata limit) public returns (uint256)
```

Functions for Modification:

```
@notice Specify the registration number for presetId. Only the sender at the time of
↳newPreset can update.
@param presetId Registration number
@param limit Set the ability value limits. Limit[6] corresponds to FOR, ABS, DFT, MND,
↳INT, EXP, in that order.
function alterPreset(uint256 presetId, Limit[6] calldata limit)
```

Values:

```
uint256 public numPresets; // Registration number, incremented and
↳assigned automatically by newPreset.
mapping(uint256 => Limit[6]) public preset; // Mapping of registration numbers to
↳ability value limit contents
mapping(uint256 => address) public presetOwner; // Mapping of registration numbers to
↳the sender at the time of newPreset
```

Limit Structure:

```
struct Limit {
    uint16 min;
    uint16 max;
}
```

46.7 Contract to Limit PERSONA Categories that Can Draw (DrawPersonaCategoryLimiter.sol)

(DrawPersonaCategoryLimiter.sol)

Set the contract in AuthorizerInfo during DrawChain registration and specify the categories you want to assign to presetId.

If the categories included in the PERSONA Id of the drawing PERSONA match the specified categories in presetId, drawing becomes possible.

46.8 Contract to Limit the Number of Draws (DrawQuantityLimiter.sol)

(DrawQuantityLimiter.sol)

Set the contract in AuthorizerInfo during DrawChain registration and specify the number of draws for presetId.

Drawing is possible if the number of draws made is less than the specified number of draws.

46.9 Contract to Limit the Caller of draw() to Subscribers of the Square Key Associated with DrawChain (DrawFollowerLimiter.sol)

(DrawFollowerLimiter.sol)

Set the contract in AuthorizerInfo during DrawChain registration.

Determine whether the user who made the draw is a follower of the square key associated with the DrawChain.

If they are a follower, drawing becomes possible

Subscribers of the square key can be placed on a blacklist.

Once on the blacklist, Subscribers will be unfollowed and cannot follow again.

To re-follow, they need to have their registration removed from the blacklist.

Registration and removal from the blacklist can be done by the owner of the square key.

SquareSupplement.sol

Functions to Register or Remove from the Blacklist:

```
@param squareKey Target square key
@param _address Follower address
@param isBlack true: Register, false: Remove from the blacklist
function setBlackList(uint256 squareKey, address _address, bool isBlack) public
```

46.10 Contract to Limit the Number of draw() Calls by the Same PERSONA (DrawCountLimiter.sol)

(DrawCountLimiter.sol)

Set the contract in AuthorizerInfo during DrawChain registration and specify the number of draws for presetId.

Drawing is possible if the number of draws made by the same persona is less than the specified number of draws.

46.11 Contract to Limit draw() Calls to Specific PERSONAs (DrawPersonaLimiter.sol)

(DrawPersonaLimiter.sol)

Specify the PERSONAs that you want to enable for draw() using newPreset.

Set the contract in AuthorizerInfo during DrawChain registration and assign the return value from the previous setup to presetId.

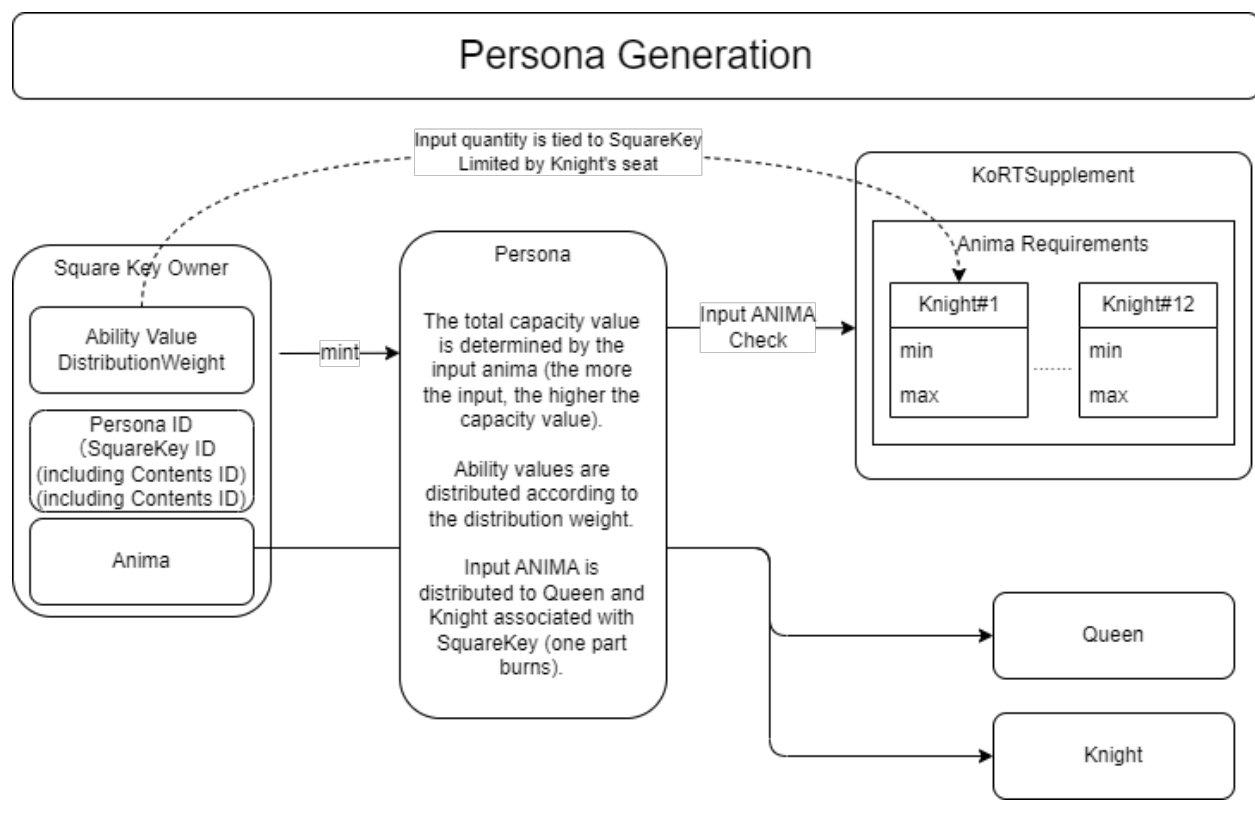
Drawing is possible if the specified PERSONA is included in the designated Preset.

newPreset:

```
@param personas Array of PERSONA IDs to be registered
@return numPresets Registration number
function newPreset(uint256[] calldata personas) public returns (uint256)
```

GENERATING AND DISTRIBUTING PERSONA

47.1 Overview Diagram



47.2 Generating PERSONA

Generating PERSONA requires consuming ANIMA. The initial attributes are determined based on the amount of ANIMA deposited during minting. A higher deposit results in higher attribute values.

The deposit amount can be set within the range specified by the seat number of the Knight to which you belong.

The distribution ratio of attributes other than FORCE can be determined by the publisher, while FORCE has a fixed ratio.

47.3 Absorbing

Refer to [here](#) for details.

47.4 PERSONA Contract

Functions for Publishers

47.4.1 Generating PERSONA (Persona.sol)

```
@param to The address where PERSONA is generated.
@param fromId The starting value of the ID portion of the minted Persona. The generated
↳ Persona token ID has the following structure for 256-bit data:
The token ID of the generated persona token has a value from fromId to fromId +
↳ numTokens - 1
All IDs must be unused.
  255                               32 31           16 15           0
+-----+-----+-----+-----+
| persona id in contents | square key | contents id |
+-----+-----+-----+-----+
@param numTokens The number of Persona tokens to mint.
@param conditions MintCondition
@return An array of generated Persona token IDs.
function mintBatch(address to, uint256 fromId, uint256 numTokens, MintCondition[]
↳ calldata conditions) public onlyMinter returns (uint256[] memory tokens)
```

47.4.2 Generating PERSONA (No Array Version) (Persona.sol)

```
function mintBatchUnified(address to, uint256 fromId, uint256 numTokens, MintCondition
↳ calldata condition) public returns (uint256[] memory tokens)
```

Determining fromId

The fromId used for minting can be obtained using the findAvailableIds function described below.

47.4.3 MintCondition

```
@param animaAmounts The amount of Anima tokens to be deposited for minting each Persona.
↳The array's length must be numTokens.
The total amount of anima in this array must be approved to the Persona contract before.
↳calling this function.
The deposit amount must be within the minimum and maximum deposit limits set for the.
↳square key's associated Knight's seat.
@param weightsList Weight allocation for the attributes of each generated Persona. The.
↳array's length must be numTokens. Each element has the following structure:
weights[n][0] Weight of ABS for the nth generated persona
weights[n][1] Weight of DFT for the nth generated persona
weights[n][2] Weight of MND for the nth generated persona
weights[n][3] Weight of INT for the nth generated persona
weights[n][4] Weight of EXP for the nth generated persona

struct MintCondition {
    uint256  animaAmounts; // Amount of anima deposited
    uint64   elements;     // Specify the element (0~6)
    uint8[5] weights;      // Weight allocation for the attributes of each generated.
↳Persona
    string   metadata;     // Set metadata
}
```

* Elements can be specified for PERSONA as described in the [lottery probability table of Elements](#).

About PERSONA's Attribute Values:

PERSONA has six attribute values **as** parameters:
 Attribute values are subject to increase **or** decrease through Absorbing.
 These attributes are also used **as** conditions **for** DrawChain execution.

```
FOR (Force/Energy)
ABS (Abyss)
DFT (Determination)
MND (Mind)
INT (Intelligence)
EXP (Experience)
```

Attribute Value Assignment:

The total attribute value is determined by the amount of anima that is put in when.
 ↳minting Persona.
 When minting a persona, the amount of anima input determines the total attribute values.
 The higher the input amount, the higher the total attribute values of the generated.
 ↳persona.
 The amount of anima input can be set within the range set by the Knight's seat number.
 Based on the total of these attribute values, each attribute value is determined.
 ↳according to the allocation weight set at the time of minting.
 Since FOR is a fixed allocation (1/6 of the total value), an allocation is set for the.
 ↳remaining attribute values.

(continues on next page)

(continued from previous page)

The maximum value for each attribute is 4195.

Example)

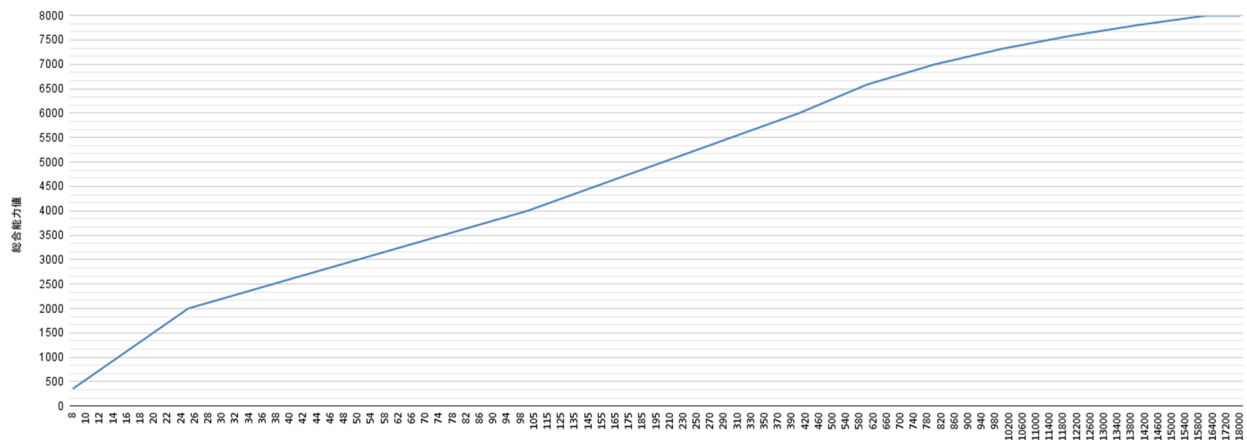
Input ANIMA amount: 10

Allocation Weight[ABS,DFT,MND,INT,EXP][2,1,1,1,1,1].

Total value: 549

Attribute value of presona generated [FOR,ABS,DFT,MND,INT,EXP][91,152,76,76,76,76].

See the following figure for the relationship between the amount of ANIMA input and the total attribute values

[illegible]

About Metadata:

Set metadata using the following steps:

Upload the desired image to IPFS and obtain the hash.

Upload a JSON file to IPFS and obtain the hash.

```
Set the obtained hash in the metadata.
```

The JSON file format should be as follows:

```
{
  "name": "persona", // Persona's name
  "creator": "user", // Creator's name
  "image": "QmYcQ3oX4M8snuesMah8cCfH5z9wuDWZm9rxLmZT5z1BzH", // Hash of the uploaded image
  "description": "" // Description
}
```


47.4.4 Setting Mutable Metadata (Persona.sol)

```
@param tokenId PersonaTokenID
@param metadata Metadata to set
function setMutableMetadata(uint256 tokenId, string memory metadata)
```

47.4.5 Getting Metadata (Persona.sol)

```
@param tokenId PersonaTokenID
@return immutableMetadata, mutableMetadata
function getMetadata(uint256 tokenId) public view returns (string memory,
↳immutableMetadata, string memory mutableMetadata)
```

47.4.6 Finding Available PERSONA IDs (Persona.sol)

```
@param _fromId Starting tokenId
@param _untilId Ending tokenId
@param numTokens Number of tokens
@return uint256 0: No IDs within the search range meet the conditions. Otherwise: The
↳first available ID.
function findAvailableIds(uint256 _fromId, uint256 _untilId, uint256 numTokens) external
↳view returns (uint256)
```

Sample Usage:

```
// Starting search value
const fromId = squareKey.shln(16);
// Ending search value
const untilId = fromId.or(new BN(
↳'ffffffffffffffffffffffffffffffffffffffffffffffffffffffff00000000', 16));
// Find available PERSONA IDs
const targetId = await persona.findAvailableIds(fromId, untilId, number of tokens to
↳search for);
// Use the searched ID for mintBatch
await persona.mintBatch(recipient's address, targetId, number of Persona tokens to mint,
↳[conditions]);
```

47.4.7 Approving the Transfer of a Specific NFT to Addresses Other Than the Owner (With Signature) (Persona.sol)

```
@param to The address to which the transfer is allowed.
@param tokenId PERSONA ID
@param nonce Refer to the signature generation procedure.
@param sig Refer to the signature generation procedure.
function approve(address to, uint256 tokenId, uint256 nonce, bytes memory sig) public
↳validToken(tokenId)
```

47.4.8 Transferring NFTs (With Signature) (Persona.sol)

```
@param from The address from which the transfer originates.
@param to The address to which the transfer is made.
@param tokenId PERSONA ID
@param nonce Refer to the signature generation procedure.
@param sig Refer to the signature generation procedure.
function transferFrom(address from, address to, uint256 tokenId, uint256 nonce, bytes_
↳memory sig) public validToken(tokenId)
```

47.4.9 Approving the Transfer of a Specific NFT to Addresses Other Than the Owner (Persona.sol)

```
@param to The address to which the transfer is allowed.
@param tokenId PERSONA ID
function approve(address to, uint256 tokenId) public validToken(tokenId)
```

47.4.10 Transferring NFTs (Persona.sol)

```
@param from The address from which the transfer originates.
@param to The address to which the transfer is made.
@param tokenId PERSONA ID
function transferFrom(address from, address to, uint256 tokenId) public_
↳validToken(tokenId)
```

USING PERSONA AS A USER

48.1 Absorbing

PERSONA can absorb ARCANA up to 5 times.

The internal values of the target determine how the internal values of the absorbed PERSONA change, and the absorbed ARCANA disappears.

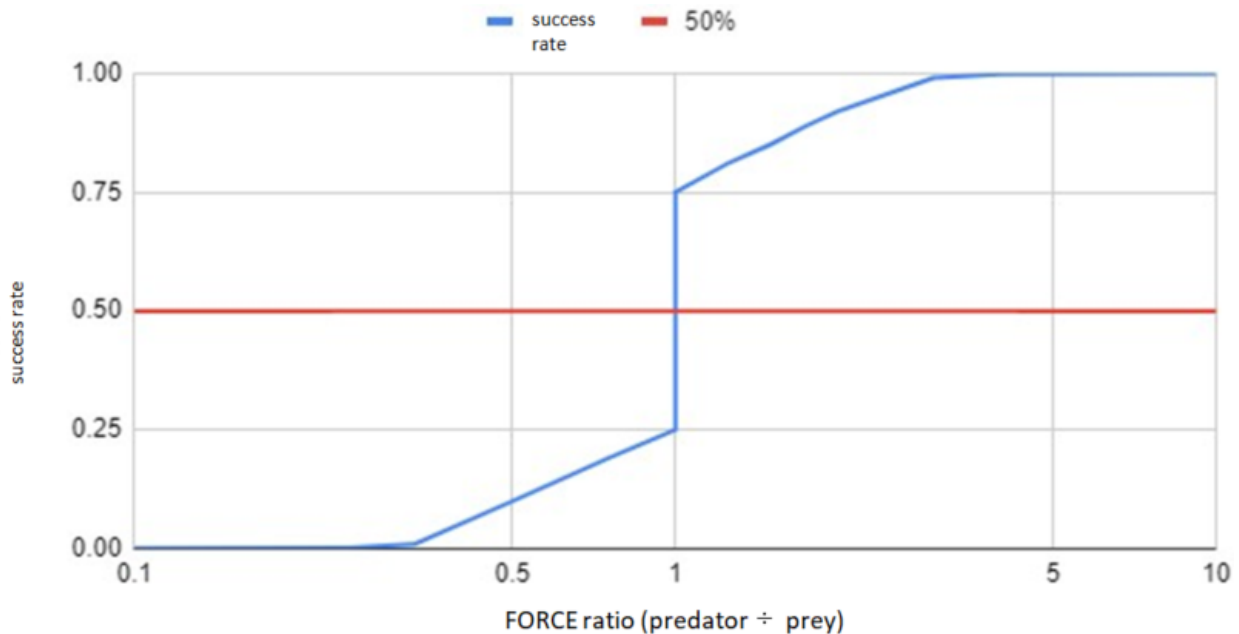
PERSONA primarily targets ARCANA with FORCE values lower than its own.

If PERSONA absorbs ARCANA with higher FORCE values, there is a higher chance of the absorbing PERSONA's internal values deteriorating.

48.1.1 • Absorbing Success Rate

The success rate of absorption is determined by comparing the total attribute values of the absorbing and absorbed parties.

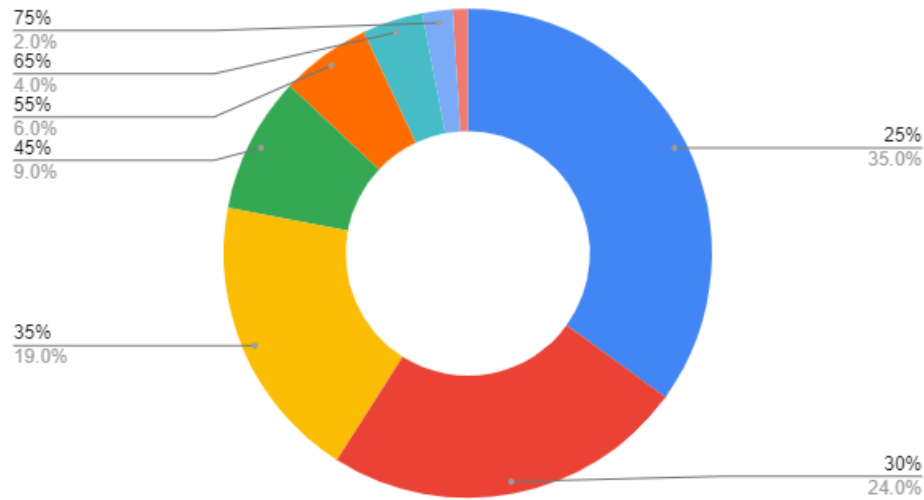
Total attribute values are calculated based on weighted internal values.



48.1.2 • Attribute Value Increase on Absorb Success

Based on the specified probabilities, a lottery is conducted for each attribute value to determine the increase.

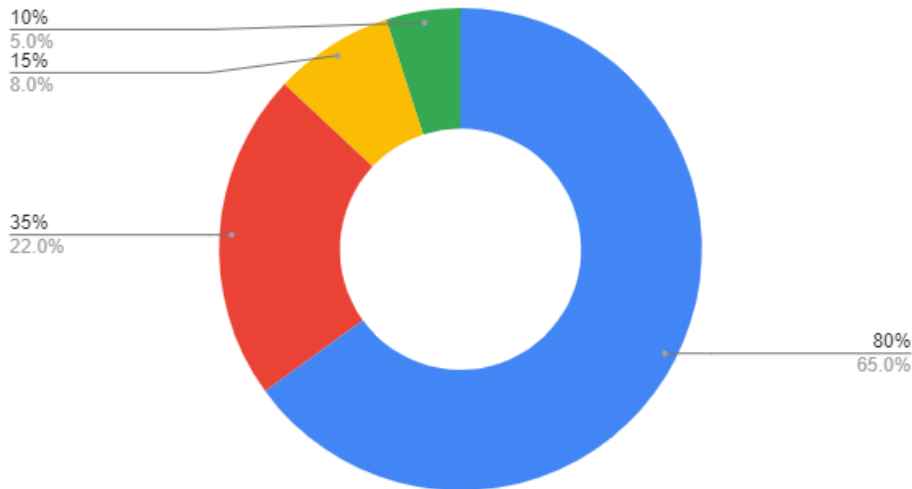
Example: There is a 35% chance of increasing the value of an absorbed opponent's attribute by 25% of its value.



48.1.3 • Attribute Value Decrease on Absorb Failure

Based on the specified probabilities, a lottery is conducted for each attribute value to determine the decrease.

Example: There is an 80% chance of decreasing the value of an absorbed opponent's attribute by 65% of its value.



About PERSONA's Attribute Values:

PERSONA has six attribute values **as** parameters:
 Attribute values can increase **or** decrease through Absorb.
 They are also used **as** conditions **for** executing DrawChain.

(continues on next page)

(continued from previous page)

FOR (Force/Energy)
 ABS (Abyss)
 DFT (Determination)
 MND (Mind)
 INT (Intelligence)
 EXP (Experience)

48.1.4 Performing Absorbing

Absorbing can be performed from the “Absorb” page of My Wallet.

Select the target PERSONA and ARCANA, then press “Absorb” to execute the process.



Functions Executed by Users During Operation

Execution of Absorb Function (Persona.sol):

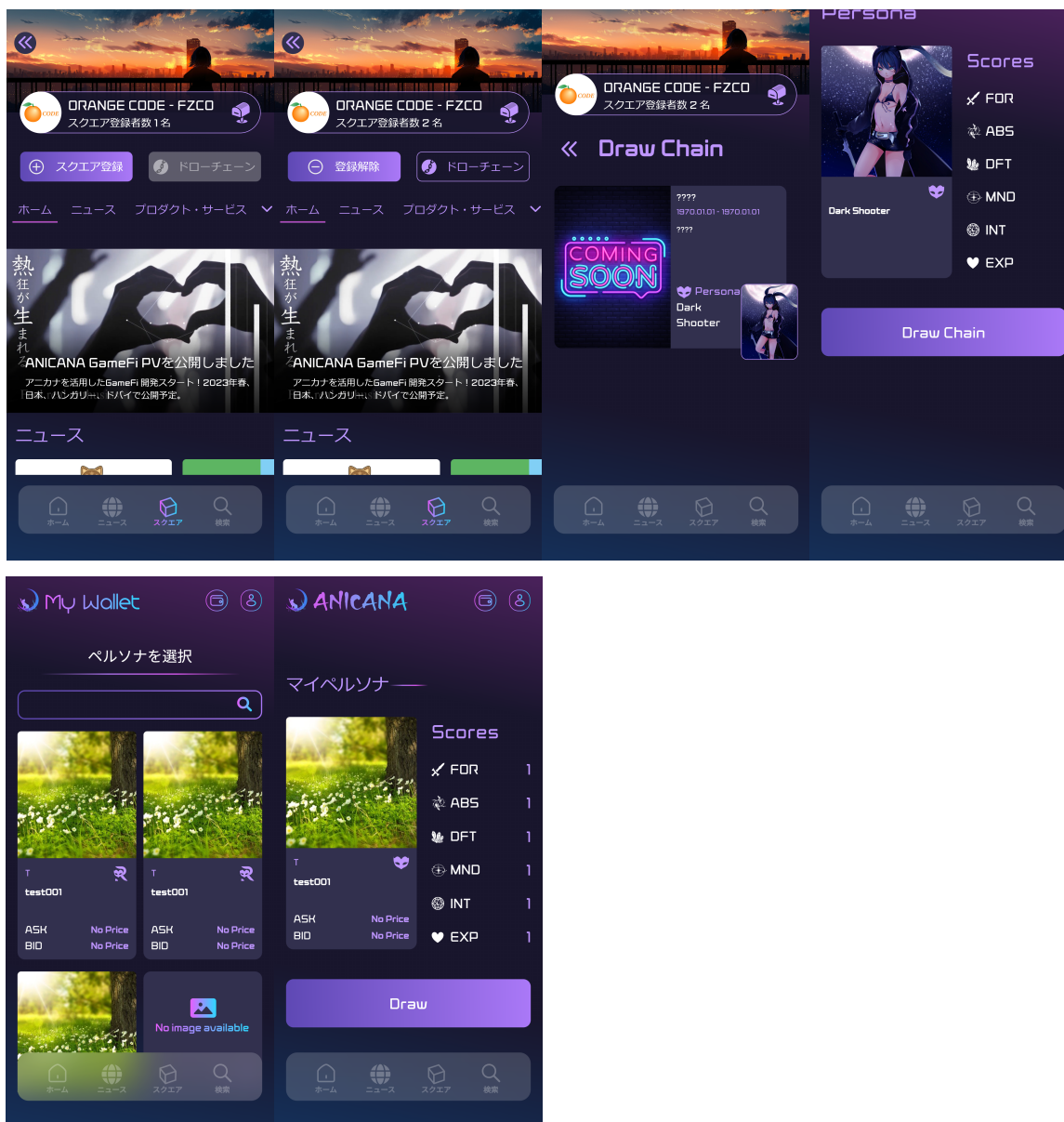
```
@param predator Predator (persona) token ID
@param prey Prey (arcana) token ID
@return true: Successful absorption. false: Failed absorption.
function absorb(uint256 predator, uint256 prey) public returns (bool)
```

48.2 Executing DrawChain

UI

Registering (subscribing) a Square is a prerequisite for using Drawchain.

Afterward, you can execute a Draw by selecting the target Drawchain and your own PERSONA.



Draw the DrawChain

Contract: Drawchain

Function Executed by Users During Operation

Execute DrawChain Function (Drawchain.sol):

```
@param drawChainId DrawChain ID
@param personaId Persona ID
@return 0: Draw failed. Non-zero: Index of history
function draw(uint256 drawChainId, uint256 personaId) public returns (uint256)
```

Have the DrawChain creator call it when distributing prizes.
Register the timestamp of the distribution (delivered).

Function for Publishers

Register Timestamp Function (Drawchain.sol):

```
@param historyId History Id returned when draw is successful
function delivered(uint256 historyId)
```

48.2.1 DrawChain Execution History

UI

Users can check their own Draw history from the “Draw History” page of the wallet.



ARCANA GENERATION INFORMATION

For details on the mechanism of ARCANA generation, please refer to [this page](#).

For environmental information, please refer to the respective environmental information pages.

49.1 ARCANA Generation Information

ARCANA generation information is managed by ArcanaGeneratorInfo and holds the following data:

InfoStatus:

```
struct InfoStatus {  
    Info info; // Information as described below  
    bool isDone; // true: generated, false: not generated  
}
```

Info:

```
struct Info {  
    string manaAddress; // Mana address  
    uint256 eggId; // Egg ID used for generation  
    address beneficiary; // Wallet address of the recipient (user)  
    uint256 seed; // Seed used for generation  
    bytes signature; // Signature used for generation  
    uint256 timestamp; // Timestamp  
}
```

Functions for Publishers

Get the length of the InfoStatus array:

```
@param beneficiary wallet address  
@returns uint256 Length of the associated InfoStatus array  
function getInfoCountByBeneficiary(address beneficiary) public view returns (uint256)
```

Get InfoStatus:

```
@param beneficiary wallet address  
@param startIndex  
@param limit Number of items to retrieve  
@returns result InfoStatus[]
```

(continues on next page)

(continued from previous page)

```
function getInfoByBeneficiary(address beneficiary, uint256 startIndex, uint256 limit)␣  
↪public view returns (InfoStatus[] memory result)
```

ANICANA API

Information about the APIs used within the Anicana service.

This page provides reference information, for the latest details, please refer to the Swagger UI in each environment.

50.1 Detail API

/detail

By specifying the token contract address in the 'category' and the token ID in 'tokenId', you can retrieve detailed data for a token.

50.2 Ipfs Upload API

/up_ipfs

This API allows you to upload files to IPFS and retrieve the uploaded hash.

AFFILIATE FUNCTIONALITY

By using the affiliate functionality, it becomes possible to track the registration and settlements of Levias IDs from the target service through referral codes.

The issuer can perform service registration on the affiliate screen and invite affiliates. Affiliates can then disseminate URLs with referral codes attached, such as through social media, and monitor the registrations and settlements made through the issued URLs on the affiliate screen.

51.1 Implementation Flow - Issuance of Referral Code

1. Issuance of Levias ID Account

* Registration is required for Levias ID accounts for invitations, proceed to register via the following link:

* stagingURL: “<https://staging.anicana.org/login/idms?f=true>”

2. Registration of Issuer Account

* Using the account registered in 1. (hereinafter referred to as the issuer account), log in and register via the issuer registration screen.

* * Please contact the ANICANA technical support team for the registration URL for the issuer account, as it changes each time.

3. Approval of Issuer Account Registration

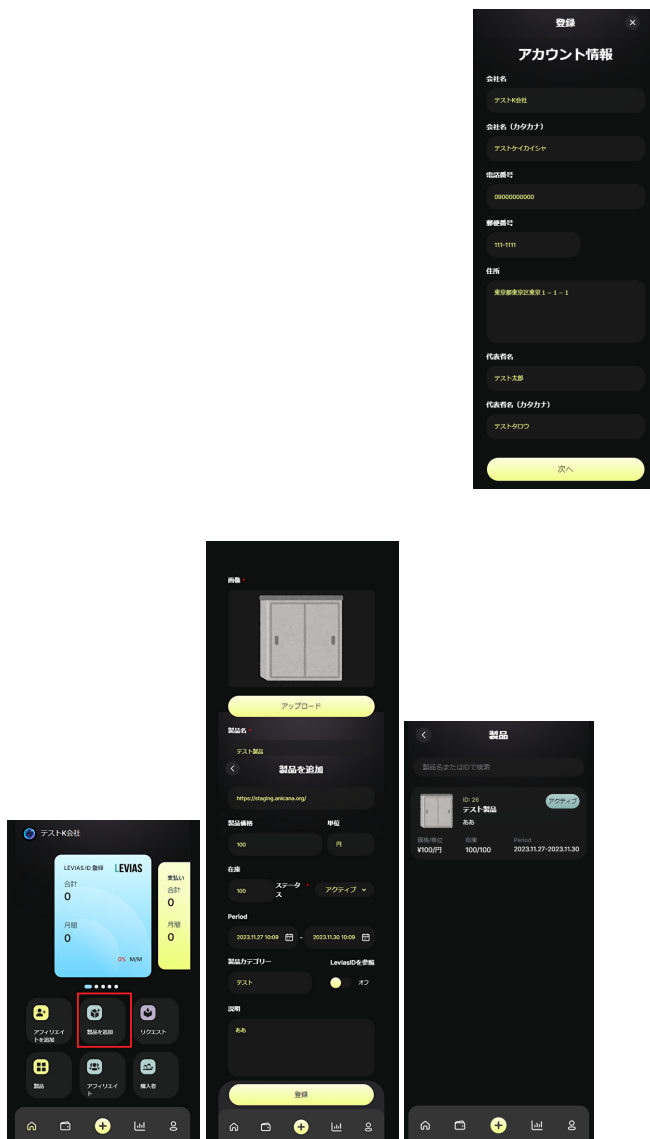
* The ANICANA technical support team approves the registration details.

* Upon approval, a confirmation email will be sent to the registered email address. Log in via the URL provided in the email.

4. Product Registration

* After approval, log in with the issuer account and proceed with product registration.

* * In the case of staging, images, URLs, etc., can be arbitrary if unavailable.



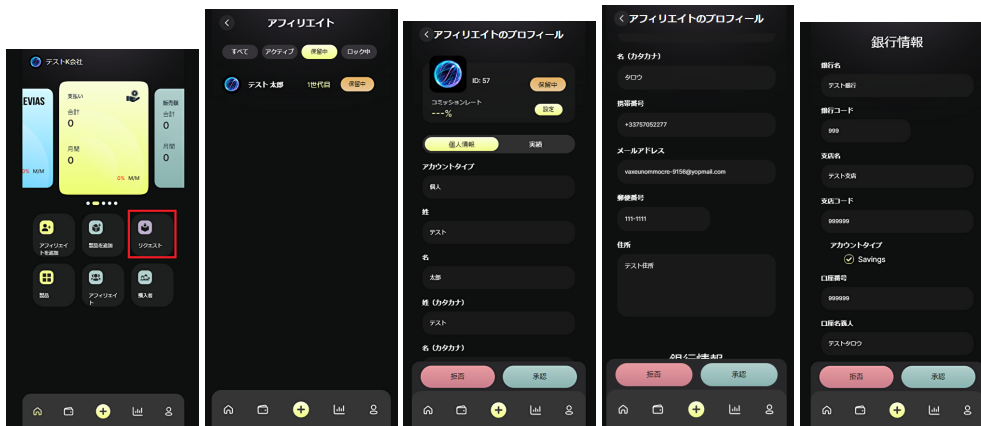
5. Affiliate Request

- * Create an affiliate after product registration.
- * Create a separate Levias ID account for the affiliate (refer to 1.).
- * Log in with the issuer account, send an invitation to the affiliate account via the affiliate addition screen.
- * Upon receiving the invitation email, log in and register with the affiliate account.



6. Approval of Request

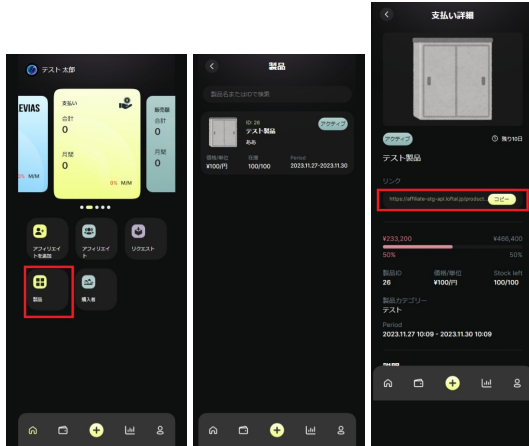
* After completing the affiliate registration, log in with the issuer account. Press the request menu and approve the affiliate.



7. Issuance of Referral Code

* Log in with the affiliate account, select the registered product, and copy the link.

* The hash at the end of the URL becomes the referral code.



51.2 Implementation Flow - Use of Referral Code

To associate registration and settlement with Levias ID, it's necessary to include the issued referral code in the registration URL and payment request.

To count Levias ID registrations, include the referral code in the generated login URL. (Counts are based on registrations, not logins)

Refer to [Wallet Connection](#).

To associate payment information, attach the referral code to Levica's Transaction Request API.

Refer to [LEVICA Payments](#).

About Aggregation

Batch aggregation occurs once daily, so numerical data for settlements, registrations, etc., will be reflected from the next day.

REQUESTING MATRIX DEVELOPMENT

52.1 Matrix Development

The EGG tokens that serve as the source for generating ARCANA tokens are created by the Matrix. Engineers can deploy their own Matrix to the network by following specific standards.

* Development engineers who have received the request will build the Matrix using the following steps.

52.2 Matrix Construction Steps

The construction involves the following steps:

- Code the Matrix according to the standards (Solidity smart contract).
 - Acquire ARCANA SHARD.
 - Deploy the Matrix contract.
 - Transfer ARCANA SHARD to the Matrix contract.
 - Set the price per EGG generation in the Matrix contract.
 - Set the content hash of metadata in the Matrix contract (refer to [Uploading to IPFS](#)).
 - Set the SquareKey ID in the Matrix contract.
 - Register the Matrix contract with the MatrixMaster contract (broadcast).
 - Inform the Validator administrator of the MatrixId assigned by MatrixMaster.
-

52.3 Matrix Standards

Refer to the [standards here](#).

52.4 Matrix Templates

Coming soon on GitHub.

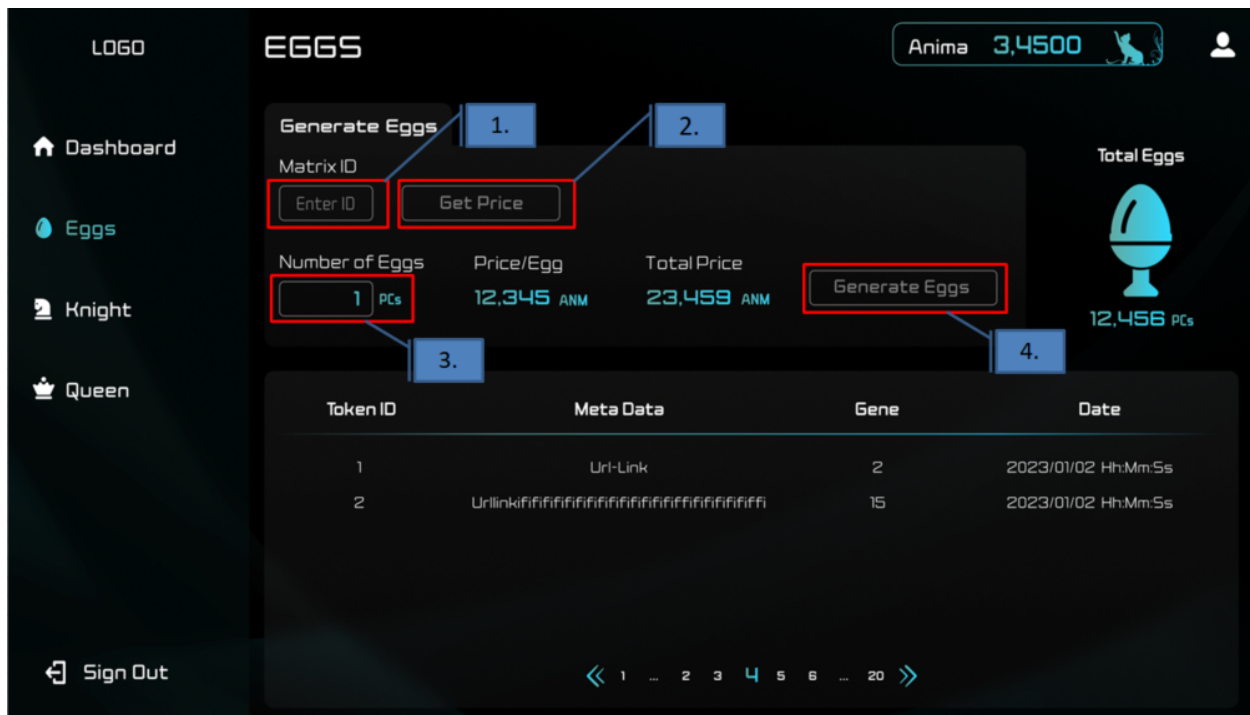
GENERATING EGGS

You can generate EGGs on the EGG page within the Validator management interface. (Validator management interfaces are deployed separately for each validator, so the URL will be different for each administrator).

[illegible]

53.1 Generation Process

1. Enter the MatrixId (ID obtained from development engineers).
2. Get the EGG generation cost with “Get Price”.
3. Input the number of EGGs to generate in “Number of eggs”.
4. Issue the EGG generation transaction with “Generate Eggs” (consumes ANIMA).



Regarding the Number of EGGs to Generate

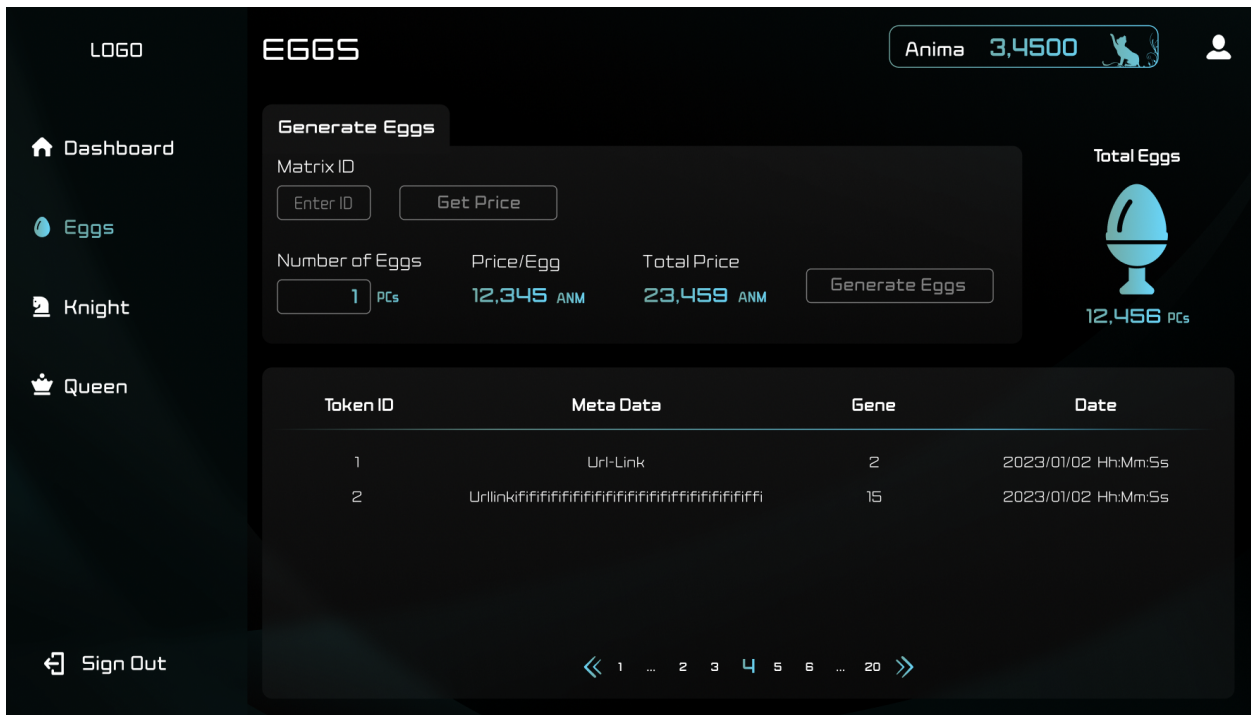
Please input a value up to a maximum of 1000 in “Number of eggs”. Values exceeding this limit may result in an error.

Checkpoints for Failed EGG Generation

- Ensure you have enough ANM for EGG generation.
- Ensure you have enough SHARD within the Matrix.
- Confirm that you possess the Square Key corresponding to the Matrix.

CHECKING EGG INVENTORY

At the bottom of the EGG screen, a list of EGG tokens owned by the user and the total number owned is displayed on the right side of the screen. Since EGGs are consumed every time a user generates an ARCANA, regular replenishment is necessary.



UPLOADING TO IPFS

* Upload the entire set of image data to IPFS for EGG conversion.

A content hash will be returned.

* Upload the metadata JSON containing the content hash of the image data to IPFS.

A content hash will be returned.

The content hash of the metadata JSON is used for [Matrix construction](#) and [Generating and Distributing PERSONA](#) .

Restrictions on images that can be uploaded

- The API to retrieve images from the PFS has the following restrictions. If the following restrictions are not met, images cannot be retrieved and therefore cannot be displayed on the portal site.
 - The file format is bmp, jpeg, png, or gif.
 - 20 MiB or less
-

55.1 Environment Information

Refer to each environment information page.

OBTAINING ANM (ANIMA)

Attention: ANM (ANIMA) is currently not listed, so there is no common method to obtain it.

56.1 What is ANIMA (ANM)?

ANIMA is the Gas token required for using ANICANA.

56.2 Instances Where ANIMA is Required as Gas

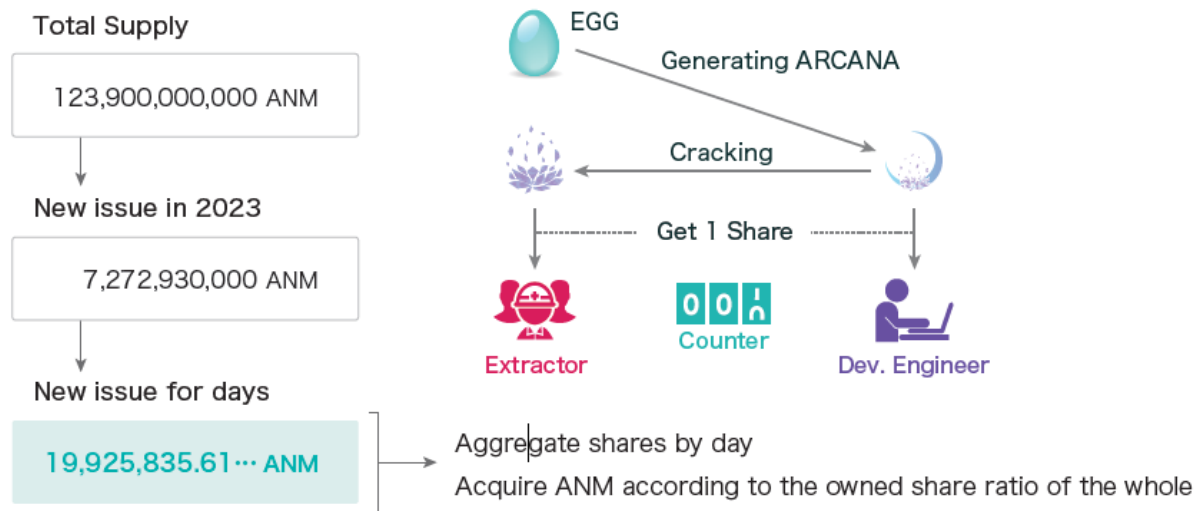
- When broadcasting MATRIX on the network.
- When generating EGGs on the ANICANA network.
- When generating PERSONAs on the ANICANA network.

Note

- Gas fees are not incurred for regular transactions.
 - Transaction gas limit is 700,000,000/tx.
 - Block gas limit is 700,000,000/block.
-

56.3 ANIMA Generation Logic

- When ARCANA is generated from EGGs, development engineers acquire 1 share.
- When ARCANA is dissolved, dissolution engineers acquire 1 share.
- The total amount of ANM generated is determined daily.
- The amount of minted ANM is determined by the shares acquired by each individual during a day.



VALIDATOR MANAGEMENT INTERFACE

Participants who are Validators or have been issued a Square Key by a Validator have access to the Validator Management Interface. It is deployed individually for each Validator, so the URL varies for each administrator. This interface allows users to apply for node participation, generate EGGs, and perform various operations related to Queens and Knights.

Preparation steps required for content development are as follows:

The Validator Management Interface is mainly used for EGG generation.

Step	Details
Validator Setup	Refer to <i>Validator Setup</i> .
Generating Square Key	Refer to <i>Generating Square Key</i> .
Requesting Matrix Development	Refer to <i>Requesting Matrix Development</i> .
Acquiring ANIMA	Refer to <i>Acquiring ANM(ANIMA)</i> .
Generating EGGs	Refer to <i>Generating EGGs</i> .

57.1 Obtaining a Private Key

You can check the Private Key of the logged-in user.

Press F12 to display the browser's development tools. Confirm the private key displayed in the console.

... figure:: ... /img/ValidatorUI/ValidatorUI-10.png

Additional details for each screen are provided below.

57.2 DASHBOARD

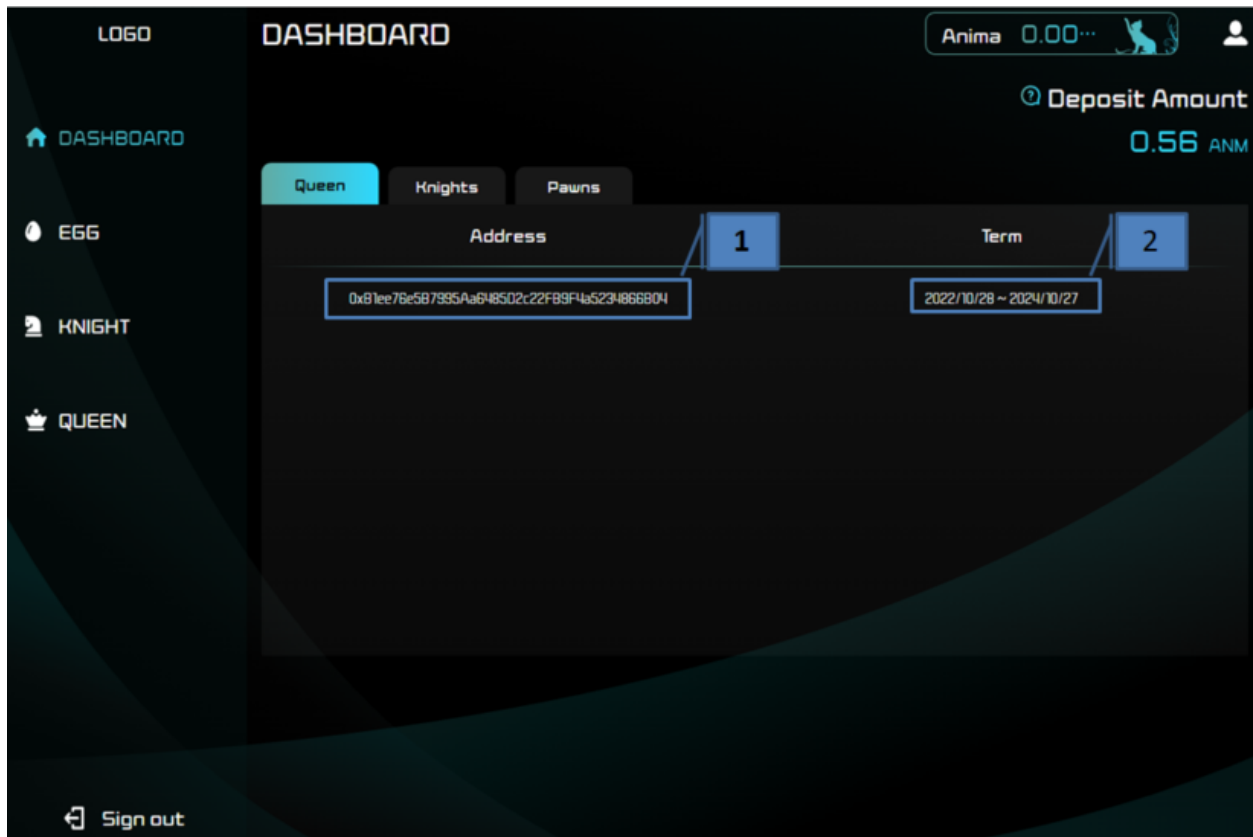
- Queen Tab

You can check the address of the Queen and her term.

1. Queen's Address
2. Queen's Term

- Knights Tab

You can check the address and number of Knights. Validators need to obtain network participation approval from one of the Knights listed here. Approved Validators are linked to the No. of the approving Knight.



1. Knight's Address
2. Knight's Number
 - Pawns



Check the participation status of other Validators. You can see the address, the No. of the approving Knight, and the joining date of the Pawn.

1. Pawn's Address
2. No. of the Knight who approved the Pawn
3. Date of joining the Pawn's node

57.3 PROFILE

Click on the icon of the user on the upper right to access the PROFILE page. Here, you can apply to be a Validator, check the status of your application, and confirm your wallet address.

1. Transition icon to PROFILE
2. Check the status of your own application
3. Apply button to Knight. Select the number of the Knight you want to apply to.
4. Your own wallet address



LOGO DASHBOARD Anima 0.00...  

Deposit Amount 0.56 ANM

Queen Knights Pawns

Address	1	Number	2
0xe7077ff38e765de9fb01ff074095fffec17a9Aa0		I	
0x529A73C85509002f80d1Fe9F48e9388F61b2B43c		II	
0x5d4738AFC00e448FCce3112D4a5c3D9EDfC85095		III	
0x26307AC507AA4aA5eC7727CD30AE6c543584b5C		IV	
0x0c58Dfb50Ce953Cef1d2a09b081774249F589156		V	
0x5ECd441EdA35B16D47D690e7CF74b21444D9f355		VI	
0xad969A6844Cfa386b0c02f84a8F5E46B48684Cfe		VII	
0x6Fe01056bEe8354b7003587bfa1A5624c7135159		VIII	
0xc6886A8917875ab2b737e0DC0Ab1f8B609F9dba		IX	
0xE675E4DEF6280F712838585Cb0A041d963947bA		X	
0xa110E018e68C01dC848a5f5E470a3A2893995FFE		XI	
0x0EA16C88e303Fa17e3eE4a9819d608Eba3CD0091		XII	

Sign out

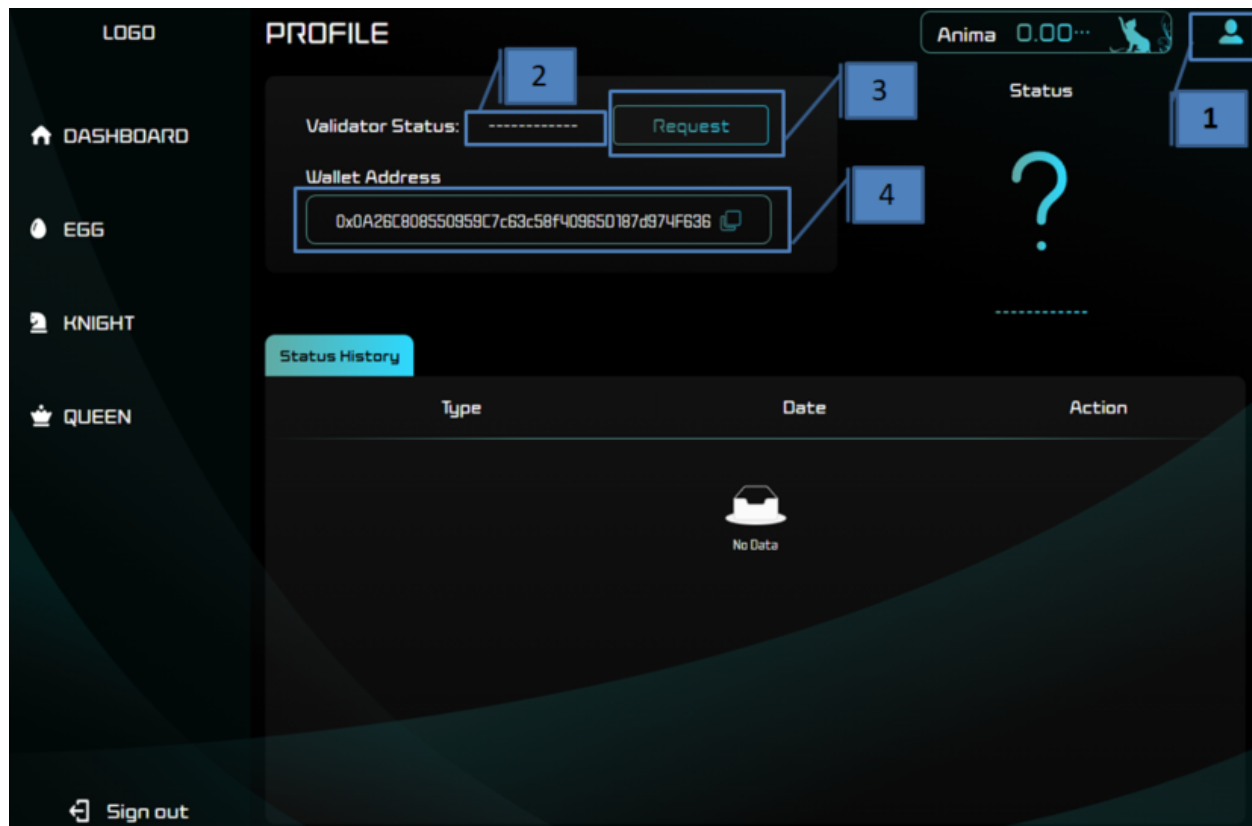
LOGO DASHBOARD Anima 0.00...  

Deposit Amount 0.56 ANM

Queen Knights Pawns

Address	1	Affiliation (Knight)	2	Date of Join	3
0xC88139a27bf26be7d7fA191c560E8A16fe489064		III		2022/11/07 15:58.12	
0xb4170e03fcf585b6028802e31f60Ab03E1e3f31		II		2022/11/02 12:29.24	
0x413510c64377dfc068389C3bd50c658d0F36064b		XI		2022/11/01 18:37.12	
0x91982905090698Ca02b3C8691993331E85FF31eE		II		2022/11/01 13:46.36	
0x6A0903235aA4E9e199b85d3F76A2427365Eede9C		XI		2022/10/31 19:36.12	
0x0Ad07C56d582C05a94521193070a08Ac0f56A025		II		2022/10/28 11:21.00	
0x91EFa38954a0b80Ec3a970520421b63cAba4c229		I		2022/10/27 08:48.24	
0x704b11845d85d55180319174338b672f6EE62597		I		2022/10/27 13:46.48	

Sign out



57.4 EGG

For information about EGG generation, refer to [here](#).

57.5 KNIGHT

You can approve new Validators, hold Queen elections, and vote for Queen removal.

1. Approve a new Validator
2. Queen election
3. Vote for Queen removal

- Pawn Requests

A list of pending approvals is displayed. You can perform approvals or denials from the list. Also, display a list of Validators approved by your Knight number.

*The list will be empty if you are not a Knight.

1. Address of the Validator who applied
2. Application date



3. Approve or Deny button

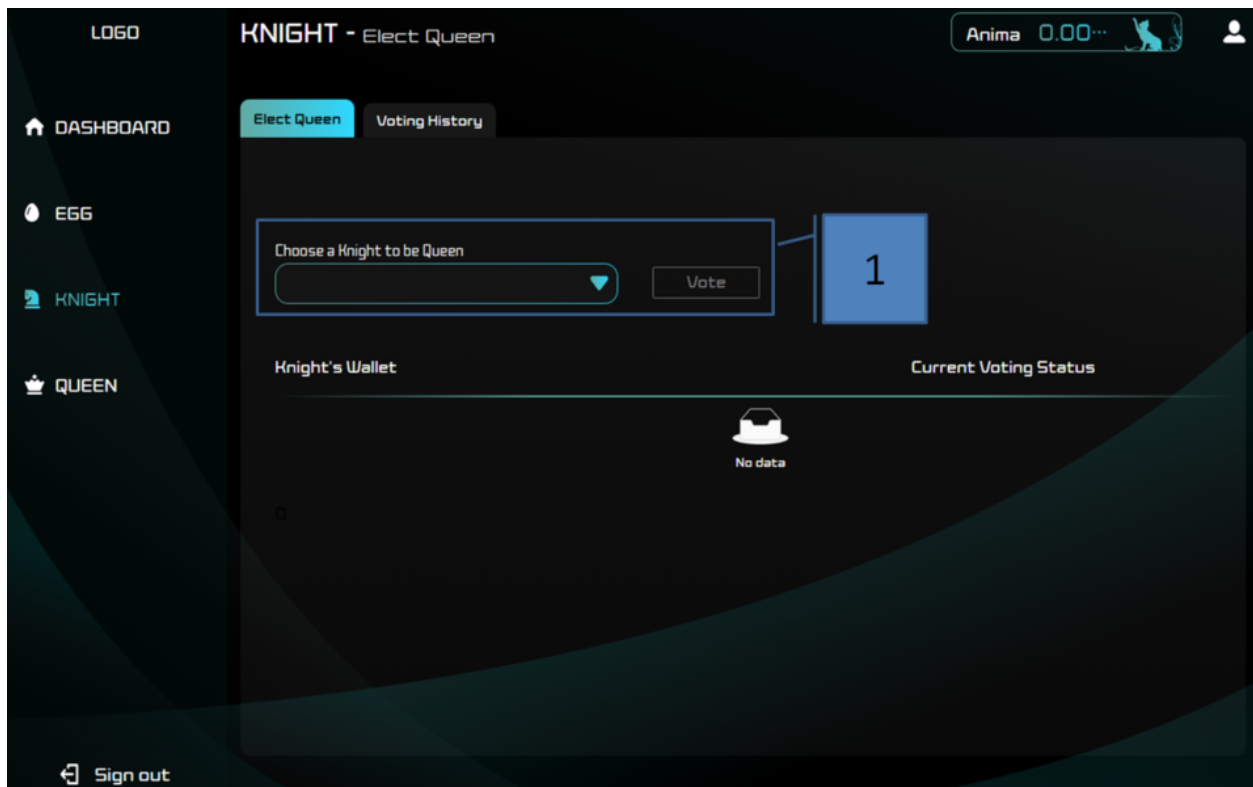
- Elect Queen

This page is used during Queen elections. If elections are in progress, you can select the voting address and cast your vote.

Also, display a list of addresses that have voted or been trusted.

※You can only view this page if you are a Knight.

1. Select a voting address and cast your vote

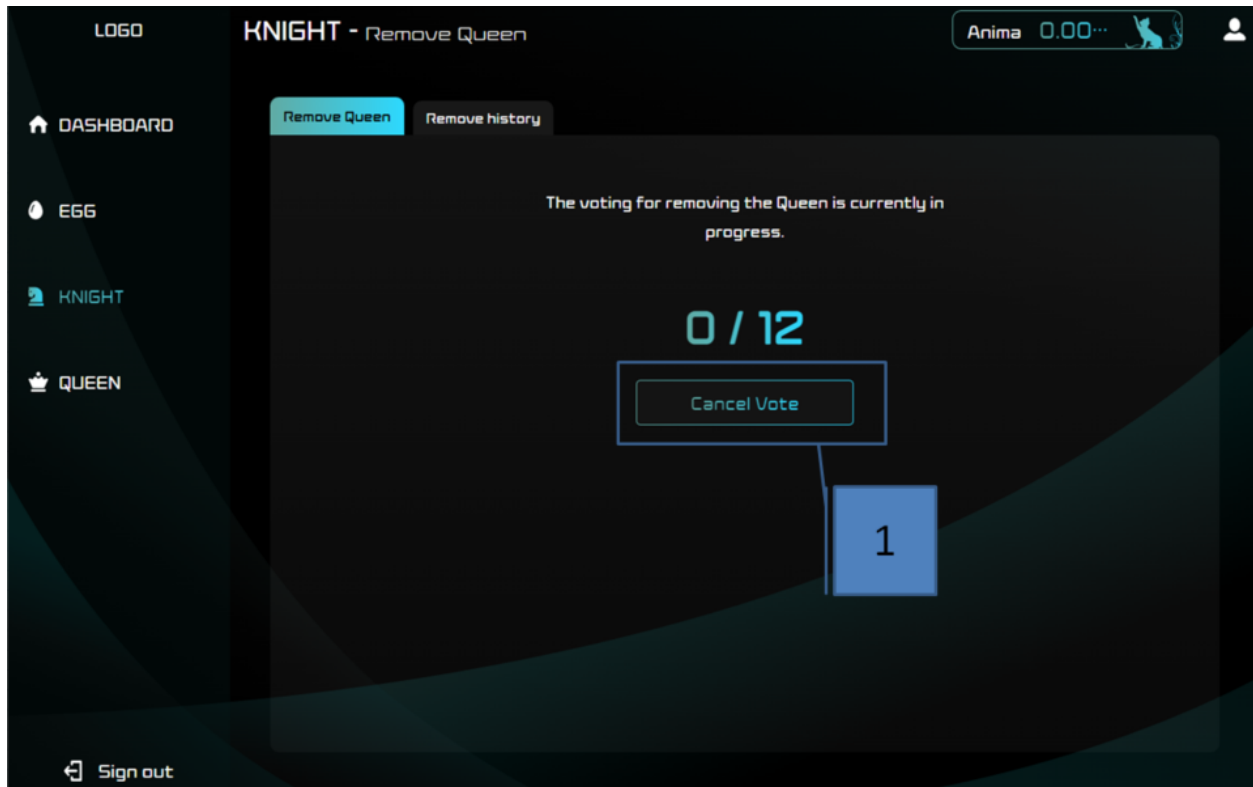


- Remove Queen

This page is used to vote for the removal of the Queen. Also, display the current number of votes.

※You can only view this page if you are a Knight.

1. Vote for Queen removal



57.6 QUEEN

This page allows operations for Queens. You can approve and remove Knights, and set the deposit amount. It also displays the current list of Knights.

※You can only view this page if you are a Queen.

1. Enter and set the ANIMA quantity
2. Remove a specific Knight. Also, specify the address and Knight number of the new Knight to appoint.

The screenshot displays the ANICANA Validator Management Interface. The top navigation bar includes a 'LOGO' placeholder, the title 'QUEEN', and a user profile section showing 'Anima' with a balance of '0.00...'. The left sidebar contains navigation links for 'DASHBOARD', 'EGG', 'KNIGHT', and 'QUEEN', along with a 'Sign out' button at the bottom.

The main content area is divided into two sections. The top section, titled 'Deposit', contains a 'Set Deposit Amount' form. A blue box labeled '1' highlights this form, which includes an 'enter amount' input field, a unit selector set to 'ANM', a display field showing '000000000000000000', and an 'Update' button. The bottom section, titled 'Knights List', features a table with columns for 'Address', 'Number', 'Date', and 'Action'. A blue box labeled '2' highlights the 'Change' button in the 'Action' column of the first row.

Address	Number	Date	Action
0xe7077f38e765de9fb01ff074095ffffec17a9Aa0	I.	2022/11/04 10:42:00	Change
0x529A73C85509002f80d1fe9f48e8388f67b2B43c	II.	2022/10/27 08:38:48	Change
0x5d4738AFC00e448FCce312D4a5c309EDfC85085	III.	2022/10/27 08:38:48	Change
0x26307AC5017AA4aA5eC7727C030AE6c543584b5C	IV.	2022/10/27 08:38:48	Change
0x0c580Fb50Ce953CeF1d2a09b08174249F589166	V.	2022/10/27 08:38:48	Change
0x5ECd441EdA35816D470690e7CF74b2144409f355	VI.	2022/10/27 08:38:48	Change
0xad969A6844Cfa386b0c02f94a8F5E46B49684Cfe	VII.	2022/10/27 08:38:48	Change
0x6Fe01056bEe8354b7003581bfa1A5624c7135169	VIII.	2022/10/27 08:38:48	Change
0xc6886A8917875ab2b737eD0C0Ab1f88609F9dba	IX.	2022/10/27 08:38:48	Change

LEVICA MERCHANT MANAGEMENT SCREEN

A screen where users can view payment histories for affiliate stores and payment information from LEVICA to affiliate stores.

58.1 Payment History Screen

User refund amounts and reuse amounts are displayed.

STORE ADMIN

ダッシュボード

決済履歴

全請求履歴

再利用設定

加盟店アカウント

サインアウト

決済履歴

決済履歴件数

100,000 件

累計

¥ 1,000,000

期間指定

日付

日付

期間指定

一括チェック

CSVファイル出力

取引番号

決済日

ユーザーLEVICA ID

決済金額

払い戻し金額

再利用金額

精算ステータス

選択

C00032	2021.12.23	B00034	¥ 50,000	¥ 50,000	¥ 50,000	精算済み	<input checked="" type="checkbox"/>
C00031	2021.07.30	B00033	¥ 51,000	¥ 51,000	¥ 51,000	未精算	<input checked="" type="checkbox"/>
C00030	2021.07.30	B00032	¥ 52,000	¥ 52,000	¥ 52,000	精算済み	<input type="checkbox"/>
C00029	2021.07.30	B00031	¥ 1,000,000	¥ 1,000,000	¥ 1,000,000	未精算	<input type="checkbox"/>
C00028	2021.00.00	B00030	¥ 999,999,999	¥ 999,999,999	¥ 999,999,999	精算済み	<input type="checkbox"/>
C00027	2021.00.00	B00029	¥ 999,999,999	¥ 999,999,999	¥ 999,999,999	未精算	<input type="checkbox"/>
C00026	2021.00.00	B00028	¥ 999,999,999	¥ 999,999,999	¥ 999,999,999	精算済み	<input type="checkbox"/>
C00025	2021.00.00	B00027	¥ 999,999,999	¥ 999,999,999	¥ 999,999,999	未精算	<input type="checkbox"/>
C00024	2021.00.00	B00026	¥ 999,999,999	¥ 999,999,999	¥ 999,999,999	精算済み	<input type="checkbox"/>
C00023	2021.00.00	B00025	¥ 999,999,999	¥ 999,999,999	¥ 999,999,999	未精算	<input type="checkbox"/>
C00022	2021.00.00	B00024	¥ 999,999,999	¥ 999,999,999	¥ 999,999,999	精算済み	<input type="checkbox"/>
C00021	2021.00.00	B00023	¥ 999,999,999	¥ 999,999,999	¥ 999,999,999	未精算	<input type="checkbox"/>

〇〇店

LEVICA ID 00001

58.2 All Billing History Screen

Total monthly sales, the refund setting upper limit amount for the following month, and the actual deposit amount are displayed. Deposit processing is done from the admin screen.

STORE ADMIN 全請求履歴 - LEVIAS社に対する、これまでの全払い戻し請求の履歴です。

〇〇店 LEVICA ID 00001

全請求履歴件数 100,000 件 累計 ¥ 1,000,000 CSVファイル出力

取引番号	対象決済金額	払戻請求設定金額	入金金額	対象期間	ステータス
C00032	¥ 50,000	¥ 20,000	---	2021.12.01 - 2021.12.31	払戻請求中
C00031	¥ 51,000	¥ 51,000	¥ 51,000	2021.07.01 - 2021.07.30	入金済
C00030	¥ 52,000	¥ 52,000	---	2021.07.30 - 2021.07.30	未入金
C00029	¥ 1,000,000	¥ 500,000	¥ 400,000	2021.07.30 - 2021.07.30	一部入金
C00028	¥ 999,999,999	¥ 999,999,999	¥ 0	2021.00.00 - 2021.00.00	再利用中
C00027	¥ 999,999,999	¥ 999,999,999	---	2021.00.00 - 2021.00.00	払戻請求設定中
C00026	¥ 999,999,999	¥ 999,999,999	¥ 999,999,999	2021.00.00 - 2021.00.00	
C00025	¥ 999,999,999	¥ 999,999,999	¥ 999,999,999	2021.00.00 - 2021.00.00	
C00024	¥ 999,999,999	¥ 999,999,999	¥ 999,999,999	2021.00.00 - 2021.00.00	
C00023	¥ 999,999,999	¥ 999,999,999	¥ 999,999,999	2021.00.00 - 2021.00.00	
C00022	¥ 999,999,999	¥ 999,999,999	¥ 999,999,999	2021.00.00 - 2021.00.00	
C00021	¥ 999,999,999	¥ 999,999,999	¥ 999,999,999	2021.00.00 - 2021.00.00	
C00020	¥ 999,999,999	¥ 999,999,999	¥ 999,999,999	2021.00.00 - 2021.00.00	
C00019	¥ 999,999,999	¥ 999,999,999	¥ 999,999,999	2021.00.00 - 2021.00.00	
C00018	¥ 999,999,999	¥ 999,999,999	¥ 999,999,999	2021.00.00 - 2021.00.00	

© LEVIAS INC. All rights reserved.

58.3 Reuse Settings Screen

ON/OFF for reuse, setting the refund upper limit amount for the next period, and setting the reuse destination wallet address.

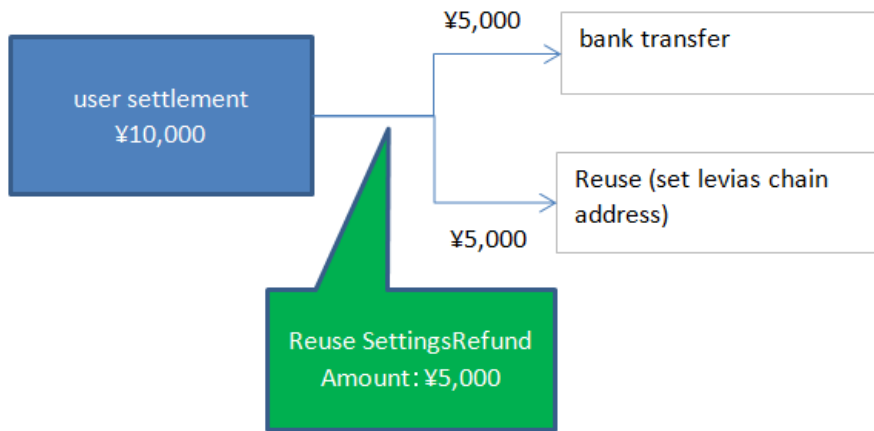
58.4 About Reuse Functionality

A feature that automatically reallocates part of the settlement to the designated levias chain address for reuse instead of refunding it via bank transfer to affiliate stores when settlements are made.

Reuse can be set by specifying the reuse amount on the management screen (Reuse Settings Screen: Figure 2) and pressing confirm after setting the destination levias chain address (Reuse Settings Screen: Figure 3).

Reuse Flow:

Reuse Settings Screen:



However, the setting timing will be for the refund upper limit amount within the range from the 11th to the next month's 10th.

Once it reaches the 11th, the setting is confirmed and cannot be changed.

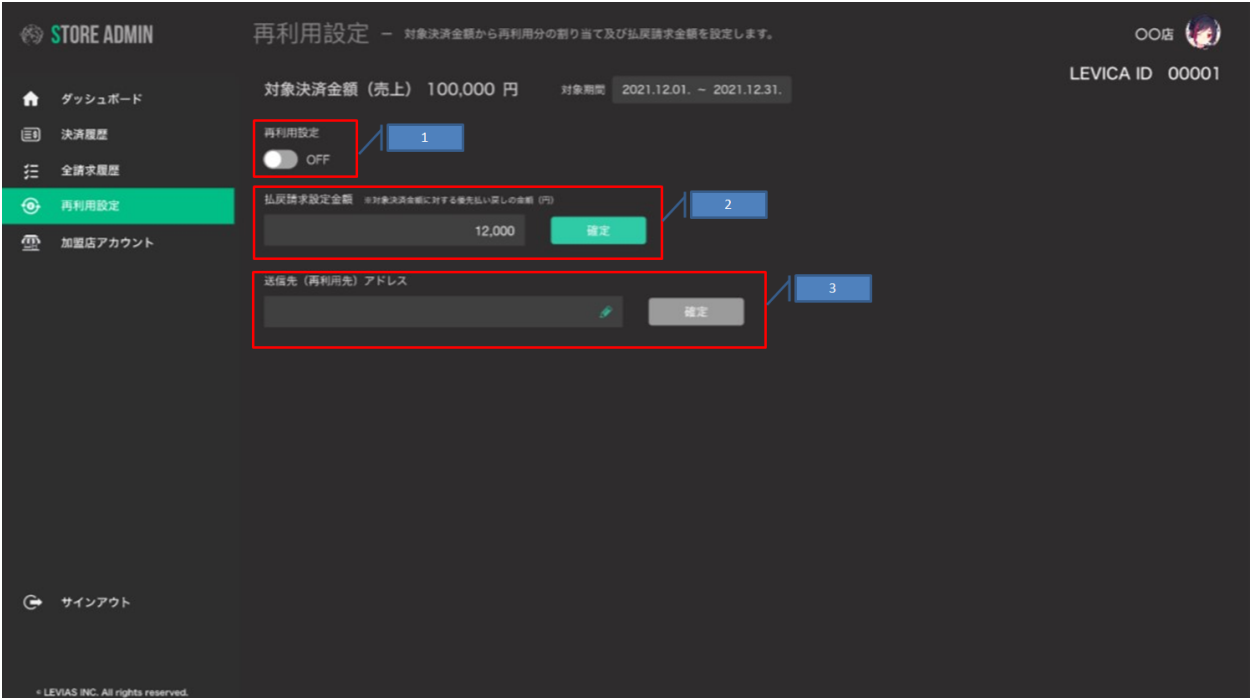
Example:

If reuse is ON, the above action will be taken. If it is OFF, everything will continue to be refunded as usual. (Reuse Settings Screen: Figure 1)

The transfer date is the end of the month. (Example: The transfer date for the refund claim period from 2/11 to 3/10 is 3/31)

If reuse is ON, it will be sent immediately to the designated levias chain address.

The app will display "Reuse" in the status.



1/11～2/10	2/11～3/10	3/11～4/10
If reuse is turned on, the reuse set by 1/10 will be executed. Reuse can be set from 2/11 to 3/10.	If reuse is turned on, the reuse set by 2/10 will be executed. Reuse can be set from 3/11 to 4/10.	If reuse is turned on, the reuse set by 3/10 will be executed. Reuse can be set from 4/11 to 5/10.

INTERFACE SPECIFICATIONS

59.1 MATRIX Standard

Contracts that have the following interfaces can be registered as a Matrix in MatrixMaster.:

```
// Provide materials of EGG to MatrixMaster.
function spawnCondition() external returns(IEggBuilder.ComposeCondition memory);

// Return the amount of ANIMA required for EGG generation. The paid ANIMA will be re
↳rewarded to the developer upon generation.
function getPrice() external view returns (uint256);

// Matrix users are limited to users with the Square Key of the returned ID here.
function correspondingSquareKey() external view returns (uint256);

// Check the owner.
function getOwner() external view returns (address);
```

59.2 Other Smart Contract Interfaces

To be published on GitHub soon.

GENE CALCULATION

60.1 Gene Overview

ARCANA, EGG, and SHARD each have unique “gene information” for every token. Gene information changes and is inherited as tokens transform from SHARD to EGG to ARCANA and back to SHARD.

Process	Gene Changes
SHARD → EGG	The gene information synthesized from two different SHARDs becomes the gene of the EGG
EGG → ARCANA	The gene information possessed by the EGG is directly inherited as the gene of the ARCANA
ARCANA → SHARD	The gene information carried by the ARCANA is directly inherited as the gene of the SHARD

60.2 Data Structure of Gene Information

Gene information is represented as a 32-dimensional vector, with each vector having a value range of 8 bits (-128 to +127). In the contract, it is stored in a single 256-bit field:

```
|-----256bit[hex]-----|
0xffeeddccbbaa99887766554433221100ffeeddccbbaa99887766554433221100
  ^^                                     ^^
Dimension 1                             Dimension 32
```

60.3 Gene Calculation for EGG Generation from SHARD

Let the gene vectors for two SHARDs X and Y, which are the basis for synthesis, be GX and GY. Let the number of X be a and the number of Y be b. The gene vector for the newly generated EGG is denoted as GZ.

The gene dimensions for the gene vectors are denoted as GXn, GYn, and GZn (n = 1 to 32).

Genes are calculated using the following formula.:

$GZ_i = (aGX_i + bGY_i) / (a+b)$
* a + b is always 100

60.4 About Mutation

Even for EGG generation between SHARDs of the same type and the same quantity, it is not guaranteed that the EGGs will have the same genes. There is a certain “mutation” logic in gene calculation.

For each of the 32 dimensions, with a 5% probability for each, the inherited value may be overwritten with a completely different new value.

ARCANA EXTRACTION

During the extraction of an ARCANA,

1. Time until extraction is complete
2. Number of resulting SHARDS

Both of these are randomly determined, and their lottery probabilities are as follows.

61.1 Duration of ARCANA Extraction

The time required for the extraction of an ARCANA token is determined through a hashing process based on the gene and salt (internal constant). The extraction time ranges from 5 minutes to 48 hours, with 16 possible values, and an expected value of 8.7 hours. The occurrence probabilities are as follows.

Duration	Probability
5.0 min	6.25%
7.6 min	6.25%
11.7 min	6.25%
17.8 min	6.25%
27.2 min	6.25%
41.6 min	6.25%
1.1 hour	6.25%
1.6 hour	6.25%
2.5 hour	6.25%
3.8 hour	6.25%
5.8 hour	6.25%
8.8 hour	6.25%
13.5 hour	6.25%
20.6 hour	6.25%
31.4 hour	6.25%
48.0 hour	6.25%

61.2 Number of ARCANA SHARDs Obtained in ARCANA Extraction

The number of SHARDs obtained when extraction an ARCANA token is determined through a hashing process based on the gene and salt (internal constant). The number of SHARDs ranges from 50 to 2500, with 16 possible values, and an expected value of 300. The occurrence probabilities are as follows.

Number Generated	Probability
50	13.91%
100	13.67%
150	12.97%
200	11.89%
250	10.53%
300	9.00%
350	7.43%
450	5.92%
550	4.56%
700	3.39%
850	2.44%
1050	1.69%
1300	1.13%
1600	0.73%
2000	0.46%
2500	0.28%

61.3 Steps for ARCANA Extraction

The extraction of ARCANA is performed following these steps.

#. Approve the ARCANA Token for the Decomposer.

Use function: `Arcana.approve(to, tokenId)`

#. Initiate the extraction.

Use function: `Decomposer.beginDecompose(tokenId)`

#. Obtain detailed information about the extraction job and confirm the completion time.

Use function: `Decomposer.getState(jobId)`

#. Wait for the extraction to be completed.

#. End the extraction and receive the SHARDs.

Use function: `Decomposer.endDecompose(jobId)`

- [Reference Video](#)

ARCANA ATTRIBUTE VALUE CALCULATION

62.1 Lottery Probability of Green Stars

The lottery for Green Star is conducted at the following probabilities when generating ARCANA.

Green Star	Probability
1	22.0%
2	19.2%
3	16.5%
4	13.7%
5	11.0%
6	8.2%
7	5.5%
8	2.7%
9	0.9%
10	0.01%

62.2 Lottery Probability of Elements

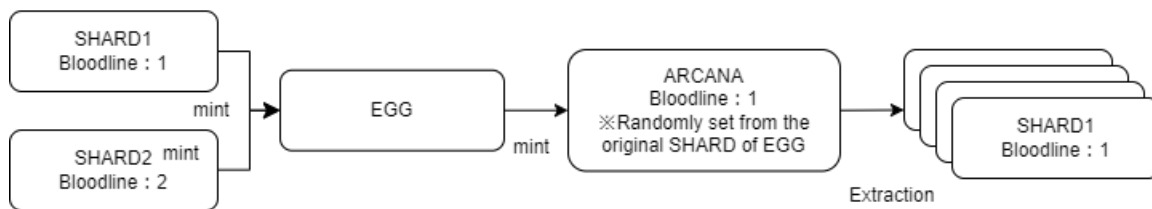
Elements are drawn with the following probabilities from the listed types.

Internal Representation	Element Name
0	Wood - Eternal Tree
1	Fire - Crimson Flame
2	Earth - Adamantine Rock
3	Metal - Peerless Steel
4	Water - Heavenly Dew
5	Light - Golden Light
6	Darkness - Unfathomable Abyss

BLOODLINE

63.1 Overview

Bloodline is the lineage to which ARCANA belongs, and every ARCANA belongs to one of the 100 Bloodlines. Each Bloodline has an ancestor, and it is inherited as shown in the diagram below, through the cycle of Egg -> ARCANA -> Shard, originating from the ancestor's Shard. Newly generated ARCANA randomly inherit one of the two Bloodlines of the Shards used in the birth Egg.



63.2 Information Retrieval

Bloodline information can be obtained by referencing the Bloodline contract. The Bloodline contract has the following interfaces.

Retrieving information about a Bloodline (Bloodline.sol):

```
// Returns the ID, name, and number of ARCANA belonging to the specified bloodline
// @param bloodlineID
// @return origin ID of the bloodline
//         nArcanas number of ARCANA belonging to it
//         name name of the bloodline
function bloodlineInfo(uint256 bloodlineID) public view returns(
    uint256 origin,
    uint256 nArcanas,
    string memory name
)
```

Retrieving the Bloodline to which an ARCANA belongs (Bloodline.sol):

```
// Returns the bloodlineID to which the ARCANA belongs using getBloodline[arcanaID]
// @param arcanaID
// @return bloodlineID
function getBloodline (uint256 arcanaID) public view returns(uint256 bloodlineID)
```

Retrieving a list of ARCANA belonging to a specific Bloodline (including already burned ARCANA)(Bloodline.sol):

```
// Returns an array of arcanaID belonging to the specified bloodlineID
// @param bloodlineID
// @param idx start index
// @param limit number of items to retrieve
// @return uint256[] memory array of arcanaID
function getBelongings(uint256 bloodlineID,uint256 idx, uint256 limit) public view
↳returns(uint256[] memory)
```

63.3 List of Bloodlines

id	name	origin
1	Zeus	Greek Mythology
2	Indra	Hinduism
3	Allah	Islam
4	Odin	Norse Mythology
5	Heracles	Greek Mythology
6	Vishnu	Hinduism
7	Hera	Greek Mythology
8	Ganesha	Hinduism
9	Osiris	Ancient Egyptian Mythology
10	Apollo	Greek Mythology
11	Persephone	Greek Mythology
12	Anubis	Ancient Egyptian Mythology
13	Loki	Norse Mythology
14	Medusa	Greek Mythology
15	Krishna	Hinduism
16	Shiva	Hinduism
17	Athena	Greek Mythology
18	Dionysus	Greek Mythology
19	Varuna	Hinduism
20	Isis	Ancient Egyptian Mythology
21	Nut	Ancient Egyptian Mythology
22	Pele	Hawaiian Mythology
23	Freya	Norse Mythology
24	Balder	Norse Mythology
25	Tezcatlipoca	Aztec Mythology
26	Izanami	Japanese Mythology
27	Ma'at	Ancient Egyptian Mythology
28	Hachiman	Shintoism
29	Pan	Greek Mythology
30	Manu	Hinduism
31	Hephaestus	Greek Mythology
32	Inti	Incan Mythology
33	Vayu	Hinduism
34	Gukumatx	Mayan Mythology
35	Frigg	Norse Mythology
36	Hestia	Greek Mythology
37	Saraswati	Hinduism

continues on next page

Table 1 – continued from previous page

id	name	origin
38	Olokun	Yoruba Mythology
39	Agni	Hinduism
40	K'an	Mayan Mythology
41	Izanagi	Japanese Mythology
42	Ta'aroa	Polynesian Mythology
43	Anu	Sumerian Mythology
44	Tyche	Greek Mythology
45	Bellona	Roman Mythology
46	Ishtar	Babylonian Mythology
47	Utu	Sumerian Mythology
48	Hecate	Greek Mythology
49	Hades	Greek Mythology
50	Set	Ancient Egyptian Mythology
51	Mithra	Persian Mythology
52	Amaterasu	Japanese Mythology
53	Thor	Norse Mythology
54	Enki	Sumerian Mythology
55	Morrigan	Celtic Mythology
56	Pachamama	Incan Mythology
57	Baron Samedi	Voodoo
58	Artemis	Greek Mythology
59	Bennu	Ancient Egyptian Mythology
60	Aphrodite	Greek Mythology
61	Ra	Ancient Egyptian Mythology
62	Brigid	Celtic Mythology
63	Tiamat	Babylonian Mythology
64	Rama	Hinduism
65	Susanoo	Japanese Mythology
66	Cronus	Greek Mythology
67	Dagda	Celtic Mythology
68	Quetzalcoatl	Aztec Mythology
69	Parvati	Hinduism
70	Bastet	Ancient Egyptian Mythology
71	Demeter	Greek Mythology
72	Fortuna	Roman Mythology
73	Narasimha	Hinduism
74	Yama	Hinduism
75	Sekhmet	Ancient Egyptian Mythology
76	Phobos	Greek Mythology
77	Lakshmi	Hinduism
78	Silvanus	Roman Mythology
79	Brahma	Hinduism
80	Nephthys	Ancient Egyptian Mythology
81	Tyr	Norse Mythology
82	Tsukuyomi	Japanese Mythology
83	Poseidon	Greek Mythology
84	Durga	Hinduism
85	Forseti	Norse Mythology
86	Eros	Greek Mythology
87	Thoth	Ancient Egyptian Mythology

continues on next page

Table 1 – continued from previous page

id	name	origin
88	Idun	Norse Mythology
89	Kali	Hinduism
90	Hermes	Greek Mythology
91	Viracocha	Incan Mythology
92	Inanna	Sumerian Mythology
93	Enlil	Sumerian Mythology
94	Ahura Mazda	Zoroastrianism
95	Janus	Roman Mythology
96	Nuada	Celtic Mythology
97	Oshun	Yoruba Mythology
98	Chaac	Mayan Mythology
99	Mictlantecuhtli	Aztec Mythology
100	Prometheus	Greek Mythology

TENEBRAE OVERVIEW

64.1 Overview

Tenebrae is a token that can be equipped on ARCANA/PERSONA and provides various effects by activating SKILL.

64.2 Issuance of Tenebrae

Tenebrae can be issued by consuming RECIPE tokens.

Depending on the type of RECIPE, the SKILL that Tenebrae can activate is determined.

64.3 Equipping Tenebrae

Up to 6 Tenebrae can be equipped on a single ARCANA/PERSONA.

Once Tenebrae is equipped, it cannot be removed.

Additionally, once Tenebrae is equipped, it cannot be transferred individually. If ARCANA/PERSONA is transferred, the equipped Tenebrae will also be transferred along with it.

64.4 Activation of Tenebrae

The effects can be obtained by activating SKILL.

SKILL has activation conditions and costs.



 *Bloodline*

Status

 *FOR* (エネルギー)

200/200

 *ABS* (深淵)

1,700/1,700

 *DFT* (意思)

199

 *MND* (精神)

2,900

 *INT* (知識)

1,000

 *EXP* (経験値)

3,189

Tenebrae 1/6

 **
Rock Crash Lv.1 >

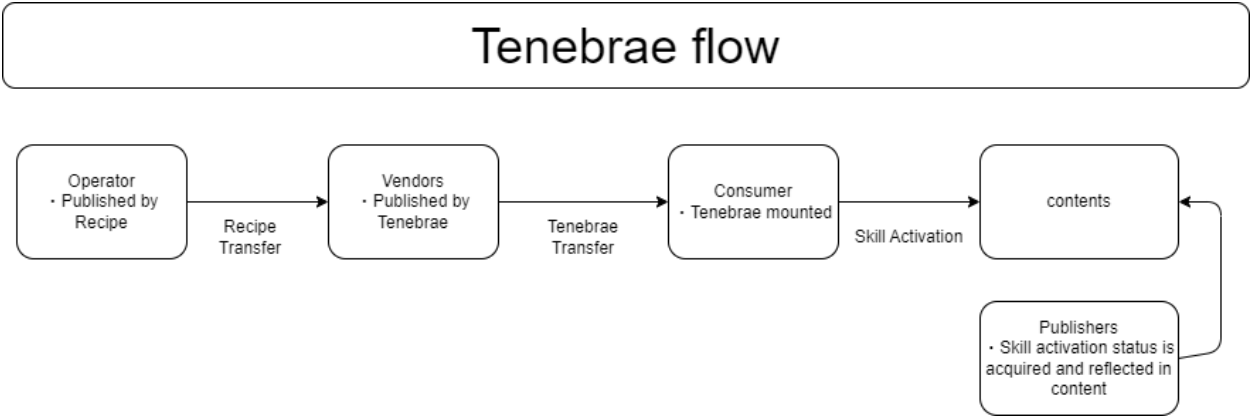
Details

売る

送る



64.5 Tenebrae Flow



The diagram illustrates the Tenebrae system architecture, showing the interactions between various components and the flow of data and ownership.

Components and Sub-components:

- Tenebrae (System):** The overall system container.
- manager:** Contains `recipeID`.
- consumer:** Contains `recipeID`, `TenebraeID`, and `ARCANAid PERSONAid`.
- RecipeController:** Contains `ITenebraeSkill list` and `ITenebraeMintAssesor list`.
- TenebraeHost:** Contains `Increase/decrease in ability value` and `List of installed TENEBRAE`.
- publisher:** Contains `ARCANAid PERSONAid`.
- manufacturer:** Contains `recipeID` and `TenebraeID`.
- Recipe:** A central component for recipe management.
- Tenebrae:** A component for Tenebrae management.
- TenebraeHistory:** A component for tracking history.

Interactions and Data Flows:

- Subscribe to list:** A solid line from the **manager** to the **RecipeController**.
- Example:** Two example boxes are shown: "Example: Spend ANM to increase the critical rate." and "Example: Generated based on a total of 100 shards of two types".
- Obtaining various lists, increase/decrease of ability values, etc.:** A solid line from the **RecipeController** to the **TenebraeHost**.
- Generate Recipe:** A solid line from the **manager** to the **Recipe**.
- Tenebrae mounting:** A solid line from the **consumer** to the **Recipe**.
- Activation of skill:** A solid line from the **consumer** to the **Tenebrae**.
- Execution history reference:** A solid line from the **consumer** to the **Recipe**.
- SKILL, conditions for generation:** A dashed line from the **RecipeController** to the **Recipe**.
- Register history:** A dashed line from the **Recipe** to the **TenebraeHistory**.
- mint:** A dashed line from the **Recipe** to the **Tenebrae**.
- Register history:** A dashed line from the **Tenebrae** to the **TenebraeHistory**.
- Skill activation information:** A dashed line from the **Tenebrae** to the **TenebraeHistory**.
- Generating Tenebrae:** A solid line from the **manufacturer** to the **Tenebrae**.
- Tenebrae Ownership Transfer:** A solid line from the **manufacturer** to the **consumer**.

65.1 Operator (RECIPE Owner)

65.1.1 Skill Contract

Create a contract implementing the ITenebraeSkill interface (ITenebraeSkill.sol):

```
@notice Activate the skill
@param tenebraeInfo Target tenebrae
@return Activation effect
function activate (TenebraeTokenInfo calldata tenebraeInfo) external returns_
↳ (ActivationEffect memory);
```

TenebraeTokenInfo[]:

```
struct TenebraeTokenInfo {
    uint256 id;                // Tenebrae ID
    uint256 attachedTo;        // Attached to ID
    uint16  activationCount;    // Number of skill activations
    uint8   attachedTokenType; // Attached to token type (TenebraeConst.TOKEN_TYPE_
↳ XXXX)
    uint8   attachedSlot;      // Attached slot number
    address skill;              // Skill
    uint256 skillSet;          // Skill set (classification within the skill)
}
```

ActivationEffect[]:

```
struct ActivationEffect {
    int16[6] abilityValues; // Changes in attached entity's ability values
    int16[6] abilityRatios; // Changes in attached entity's ability ratios
    bool[6]  resetAbility;  // true: reset ability changes
    bool      dismissal;    // true: detach Tenebrae after execution. false: don't_
↳ detach from slot
    string    skillMetadata; // Metadata to be set as activation result
}
```

TenebraeConst:

```
contract TenebraeConst {
    uint8 public constant TOKEN_TYPE_NONE = 0;
    uint8 public constant TOKEN_TYPE_SHARD = 1;
    uint8 public constant TOKEN_TYPE_ARCANA = 2;
    uint8 public constant TOKEN_TYPE_PERSONA = 3;
    uint8 public constant TOKEN_TYPE_EGG = 4;

    uint8 public constant EVENT_MINT = 0;
    uint8 public constant EVENT_ATTACH = 1;
    uint8 public constant EVENT_ACTIVATE = 2;
    uint8 public constant EVENT_VANISH = 3;
}
```

65.1.2 Registration of Skill Contract

Register a contract implementing the ITenebraeSkill interface (RecipeController.sol):

```
@notice Register TenebraeSkill
@param assesor // Contract address implemented
@param description // Description
@return skillId
function registerSkill(address skill,string calldata description) public override_
↳onlyRegisterer returns (uint256)
```

65.1.3 Reference of Skill Contract

Reference a contract implementing the ITenebraeSkill interface (RecipeController.sol):

```
@param skillId
@return RegisteredInfo
function getSkill(uint256 skillId) public override view returns (RegisteredInfo memory)
```

For multiple:

```
@param offset @param limit @return RegisteredInfo function getSkills(uint256 offset,uint256 limit) public override view returns (RegisteredInfo[] memory)
```

Get the number of registered skills:

```
function getLastSkillId() public override view returns (uint256)
```

65.1.4 Tenebrae Mint Assessor Contract

Create a contract implementing the ITenebraeMintAssesor interface (ITenebraeMintAssesor.sol):

```
@notice Determine whether the mint conditions of Tenebrae are met.
@param tokenType Type of token TenebraeConst.TOKEN_TYPE_XXXX
@param ids Array of token IDs to consume
@param amounts Quantities of tokens to consume
@return true: mintable under specified conditions, false: not mintable under specified_
↳conditions
function asses(uint8 tokenType,uint256[] calldata ids,uint256[] calldata amounts)_
↳external view returns (bool);
```

65.1.5 Registration of Tenebrae Mint Assessor Contract

Register a contract implementing the ITenebraeMintAssesor interface (RecipeController.sol):

```
@notice Register TenebraeMintAssesor
@param assesor // Contract address implemented
@param description // Description
@return assesorId
function registerAssesor(address assesor,string calldata description) public override_
↳onlyRegisterer returns (uint256)
```

65.1.6 Reference of Tenebrae Mint Assessor Contract

Reference a contract implementing the ITenebraeMintAssesor interface (RecipeController.sol):

```
@param assesorId
@return RegisteredInfo
function getAssesor(uint256 assesorId) public override view returns (RegisteredInfo,
↳memory)
```

For multiple:

```
@param offset @param limit @return RegisteredInfo function getAssesors(uint256 offset,uint256 limit)
public override view returns (RegisteredInfo[] memory)
```

Get the number of registered assessors:

```
function getLastAssesorId() public override view returns (uint256)
```

65.1.7 Recipe Minting

Mint a Recipe (Recipe.sol):

```
@notice mint
@param to Address of the recipient
@param skillId Skill ID to grant
@param assesorId Assesor ID of minting conditions
@return ID of the minted Tenebrae token
function mint(address to,uint256 skillId,uint256 skillSet,uint256 assesorId) public,
↳onlyMinter returns (uint256)
```

TokenInfo:

```
struct TokenInfo {
    uint256 id;           // RECIPE ID
    address assesor;     // Assesor contract for Tenebrae MINT conditions
    address skill;       // Skill
    uint256 skillSet;    // Classification within the skill
}
```

65.1.8 Fetching Ability Values of ARCANA Token/PERSONA Token

Get the incremental changes in ability values (TenebraeHost.sol):

```
@param hostType Type of TenebraeConst.TOKEN_TYPE_XXXX
@param hostId Host ID
@return HostInfo
function hostInformation(uint8 hostType,uint256 hostId) public view returns (HostInfo,
↳memory)
```

HostInfo:

```
/// @notice Holds the changes in ability values
struct HostInfo {
```

(continues on next page)

(continued from previous page)

```

    /// @notice ARCANA/PERSONA
    uint8 hostType;
    /// @notice tokenId of ARCANA/PERSONA
    uint256 hostId;
    /// @notice slot for attaching TENEBRAE
    uint256[] slot;
    /// @notice metadata of the result of activation of TENEBRAE
    string[] activatedMetadata;
    /// @notice increments of attribute values
    int32[6] incrementValues;
    /// @notice multiplier of attribute values (1/100000)
    int32[6] incrementRatios;
}

```

Get the ability values of Arcana (original + adjusted values) (TenebraeHost.sol):

```

@notice Get the ability values of Arcana (original + adjusted values).
@param tokenId Token ID of Arcana
@return original Arcana's original ability values
@return currentAbilities Adjusted ability values
function getArcanaParameters(uint256 tokenId) external view returns (IArcana.Parameters,
    ↪memory original,uint16[] memory currentAbilities)

```

Get the ability values of Persona (original + adjusted values) (TenebraeHost.sol):

```

@notice Get the ability values of Persona (original + adjusted values).
@param tokenId Token ID of Persona
@return original Persona's original ability values
@return currentAbilities Adjusted ability values
function getPersonaParameters(uint256 tokenId) external view returns (uint16[] memory,
    ↪original,uint16[] memory currentAbilities)

```

65.1.9 Fetching the List of Equipped TENEBRAE

Get the list of equipped TENEBRAE (TenebraeHost.sol):

```

@notice Get the list of TENEBRAE attached to ARCANA/PERSONA
@param hostType Type of target ARCANA/PERSONA TenebraeConst.TOKEN_TYPE_XXXX
@param hostId ID of target ARCANA/PERSONA
@return Attachment status of attachment slots (0 indicates not attached)
function getAttached(uint8 hostType,uint256 hostId) public view validType(hostType)
    ↪returns (uint256[] memory)

```

65.1.10 Fetching History

Get history (TenebraeHost.sol):

```
@param tenebraeId Tenebrae ID
@return history
@dev Get history
function getHistory(uint256 tenebraeId) public override view returns(History[] memory)
```

TokenInfo:

```
struct History {
    uint8   eventType;      // TenebraeConst.EVENT_XXX
    uint64  timestamp;
    address triggeredBy;    // Executor address msg.sender
}
```

65.1.11 Granting and Revoking Access to Ability Value Modification Functions

Grant access (TenebraeGameIF .sol):

```
function grantAccess(address _addr) public onlyAuthority
```

Revoke access (TenebraeGameIF .sol):

```
function revokeAccess(address _addr) public onlyAuthority
```

65.2 Manufacturer

65.2.1 Production of TENEBRAE Tokens

Mint TENEBRAE (Recipe.sol):

```
@param recipeId
@param mintTo
@param shardIds // IDs of consumed shards
@param amounts  // Quantities of consumed shards
function produceByShard(uint256 recipeId,address mintTo,uint256[] calldata shardIds,
    ↪uint256[] calldata amounts) public onlyOwner validToken(recipeId) returns (uint256)
```

65.3 Consumer (TENEBRAE Token Owner)

65.3.1 Equipping TENEBRAE Tokens

Attach TENEBRAE to ARCANA/PERSONA (TenebraeHost.sol):

```

@notice Attach TENEBRAE to ARCANA/PERSONA
@param hostType Type of target ARCANA/PERSONA TenebraeConst.TOKEN_TYPE_XXXX
@param hostId ID of target ARCANA/PERSONA
@param tenebraeId Target attachment
@return Index of the attached slot (0-based)
@dev Revert with revert message E10 if there are no available slots
function attach(uint8 hostType,uint256 hostId,uint256 tenebraeId) public override
↳ returns (uint256)

```

65.3.2 Activation of Skill

Activate Skill (TenebraeToken.sol):

```

@notice active
@param tenebraeId Target TENEBRAE
function activate(uint256 tenebraeId)

```

65.4 Publisher

65.4.1 Fetching Metadata Set by Activating TENEBRAE

Get the list of activation data of SKILL (TenebraeHost.sol):

```

@notice Get the list of activation data of SKILL.
@param hostType Type of target ARCANA/PERSONA TenebraeConst.TOKEN_TYPE_XXXX
@param hostId ID of target ARCANA/PERSONA
@return List of skill activation data
function getActivatedSkills(uint8 hostType,uint256 hostId) external view
↳ validType(hostType) returns (ActivatedSkillInfo[] memory)

```

Specify slot (TenebraeHost.sol):

```

@notice Get the activation data of SKILL
@param hostType Type of target ARCANA/PERSONA TenebraeConst.TOKEN_TYPE_XXXX
@param hostId ID of target ARCANA/PERSONA
@param slotIdx Index of target attached slot
@return Skill activation data
function getActivatedSkill(uint8 hostType,uint256 hostId,uint8 slotIdx) external view
↳ validType(hostType) returns (ActivatedSkillInfo memory)

```

ActivatedSkillInfo:

```

/// @notice Metadata information of the activated skill
struct ActivatedSkillInfo {
    /// @notice Slot index of attachment
    uint8 slotIdx;
    /// @notice Metadata of the activated SKILL
    string metadata;
}

```

65.4.2 Consuming Metadata Set by Activating TENEBRAE

Consume activation data of SKILL (TenebraeHost.sol):

```
@notice Consume activation data of SKILL (delete - clear).
@param hostType Type of target ARCANA/PERSONA
@param hostId ID of target ARCANA/PERSONA
@param slotIdx Index of target attached slot
function consumeActivatedData(uint8 hostType,uint256 hostId,uint8 slotIdx) public
↳validType(hostType)
```

65.4.3 Fetching Ability Value Changes as Game Results

Reference Operator (RECIPE Owner)

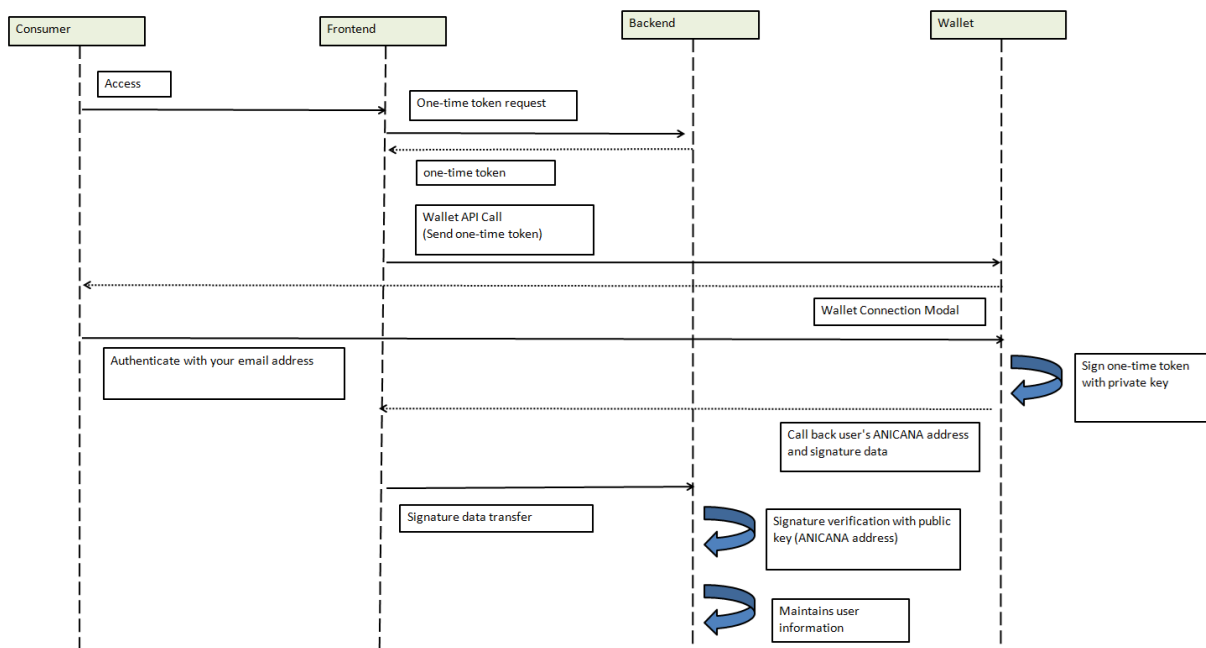
65.4.4 Updating Incremental Values and Ratios of ARCANA/PERSONA Ability Values (Authorization Required)

Consume activation data of SKILL (TenebraeHost.sol):

```
@param hostType Type of target ARCANA/PERSONA
@param hostId ID of target ARCANA/PERSONA
@param values Values to set in HostInfo's incrementValues
@param ratios Values to set in HostInfo's incrementRatios
function updateAbilities(uint8 hostType,uint256 hostId,int16[6] calldata values,int16[6]
↳calldata ratios) public onlyGranted
```


ADVANCED SECURITY SETTINGS FOR WALLET CONNECTION

Through the ANICANA Wallet Server, it is possible to obtain a user's address from their content and verify that they are the owner of that address. Below is an example procedure.



Consumer: User of the content
Frontend: Content's frontend
Backend: Content's backend
Wallet: ANICANA Wallet Server

66.1 One-Time Token

On the wallet side, simply sign the given text with the user's private key and return it. The signing method used here is the Elliptic Curve Digital Signature Algorithm (ECDSA). Handling this signed data is delegated to the consumer (content) system. To verify if the owner of the address sent in the callback after login is indeed the user in question, you need to verify this signed data. Since the signer's address can be obtained from the signed data and the message before signing, this can be used as a means of confirming the identity of the declared address. To prevent the reuse of signed data, it is advisable to use disposable target text.

Caution: Please use web3 version 1.9.8.

Example of signature creation:

```
var Web3 = require('web3');
var web3 = new Web3("https://RPC_ENDPOINT");

var original_message = "Hello world";
var privateKey = "0xYOURPRIVATEKEYXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX";
var signedData = web3.eth.accounts.sign(original_message, privateKey);
```

Example of signedData:

```
{
  message: 'Hello world',
  messageHash: '0x8144a6fa26be252b86456491fbcd43c1de7e022241845ffea1c3df066f7cfede',
  v: '0x1b',
  r: '0x399ab420d35d6d40e55580317b5fbb907942b6e35f56c22ddd306bd7b13aef8d',
  s: '0x4cedaa39073a5e626043228a20d2a386d9e0a80f5cafb90ac0798559b7b82d1d',
  signature:
    ↪ '0x399ab420d35d6d40e55580317b5fbb907942b6e35f56c22ddd306bd7b13aef8d4cedaa39073a5e626043228a20d2a386d9e0a80f5cafb90ac0798559b7b82d1d'
    ↪
}
// signature is the signed data
```

Verification example:

```
var Web3 = require('web3');
var web3 = new Web3("https://RPC_ENDPOINT");

var original_message = "Hello world"
var signature = "XXXXXXXXXXXXXXXXXXXX" // Signature data sent in the callback
var signer = web3.eth.accounts.recover(original_message, signature);
```

Example of signer:

```
0x7E99a37fFc1D9eCC05C9ac0c65598F8215c01582
```


UPDATE HISTORY

No	Date of Update	Version	Update Details
26.	2024/05/01	1.5.1	LEVICA API (Production) LEVICA API (Ark.one) LEVIAS ID (Production) LEVIAS ID (Ark.one) Update of the release contents
25.	2024/04/26	1.5.0	tenebrae overview tenebrae Implementation Create page
24.	2024/04/21	1.4.8	Bloodline Update the list of Bloodline
23.	2024/04/10	1.4.7	Production Environment Information Added Boloodline contract addresses and abi Bloodline Update the list of Bloodline
21.	2024/04/03	1.4.6	LEVICA API (Production) Update scheduled release date
198	2024/03/28	1.4.5	