Security Review Report
NM-0612 - Regeneration Credit

(October 04, 2025)

Contents

1 Executive Summary **2**

2 Audited Files **3**

3 Summary of Issues **4**

4 Protocol Overview **5**
4.1 Actors and Their Roles . . . 5
4.2 Structure of the Contracts and Interaction with Users . . . 5

5 Risk Rating Methodology **7**

6 Issues **8**
6.1 [High] Fraud Reporting Mechanism Fails to Protect Protocol Integrity . . . 8
6.2 [High] Inspectors Are Unfairly Penalized When Detecting Invalid Regenerators . . . 11
6.3 [High] Inspectors Can Inflate Ecological Value to Profit from Token Sales . . . 13
6.4 [High] Insufficient validation of coordinates can lead to double registration of a regeneration area . . . 15
6.5 [Medium] Any type of user can accept invitation from denied users . . . 16
6.6 [Medium] Risk of Fake Denunciation Undermining User Reputation System . . . 17
6.7 [Medium] Unexecuted Inspections may Block Regenerator Rewards . . . 19
6.8 [Low] The average level decreases when denied users are included in the vote power computation . . . 21
6.9 [Info] Accumulation of Residual Tokens in Pools . . . 22
6.10 [Info] Imbalanced Token Distribution in Inspector Pool Reduces Incentives . . . 23
6.11 [Info] Inspectors are deactivated in their three give-ups . . . 23
6.12 [Info] Lack of Incentives for Inspection Invalidation . . . 24
6.13 [Info] Permanent Pool Misconfiguration Risk After Ownership Renouncement . . . 27
6.14 [Info] Unallocated tokens in `Eras` and `Epochs` will be completely locked in the contract . . . 27
6.15 [Info] Unclaimed Rewards Across Multiple Eras May Become Unrecoverable . . . 28
6.16 [Info] _canVoteRules function returns true when totalUsers is zero . . . 29
6.17 [Info] `canInvite` function returns true when `totalUsers` is zero . . . 30
6.18 [Best Practice] Event in `_denyUseris` never emitted . . . 31

7 Documentation Evaluation **32**

8 Test Suite Evaluation **33**
8.1 Tests Output . . . 33

9 About Nethermind **65**

1 Executive Summary

This document presents the results of a security review conducted by Nethermind Security for Regeneration Credit protocol. It establishes a funding system that directly rewards those engaged in the restoration of ecosystems.

Basically, the Regeneration Credit enables regenerators to earn additional income by transforming measurable ecological impact into digital credits while contributing to the recovery of biodiversity and natural resources. These credits can be converted into tokens that can be exchanged in the open market.

The audit comprises 3286 lines of Solidity code. **The audit was performed using** (a) manual analysis of the codebase, and (b) automated analysis tools.

Along this document, we report 18 points of attention where four are classified as `High`, three are classified as `Medium`, one as `Low` and ten are classified as `Informational` or `Best Practices`. classified as `Informational` severity. The issues are summarized in Fig. 1.

This document is organized as follows. Section 2 presents the files in the scope. Section 3 summarizes the issues. Section 4 presents the system overview. Section 5 discusses the risk rating methodology. Section 6 details the issues. Section 7 discusses the documentation provided by the client for this audit. Section 8 presents the test suite evaluation and automated tools used. Section 9 concludes the document.

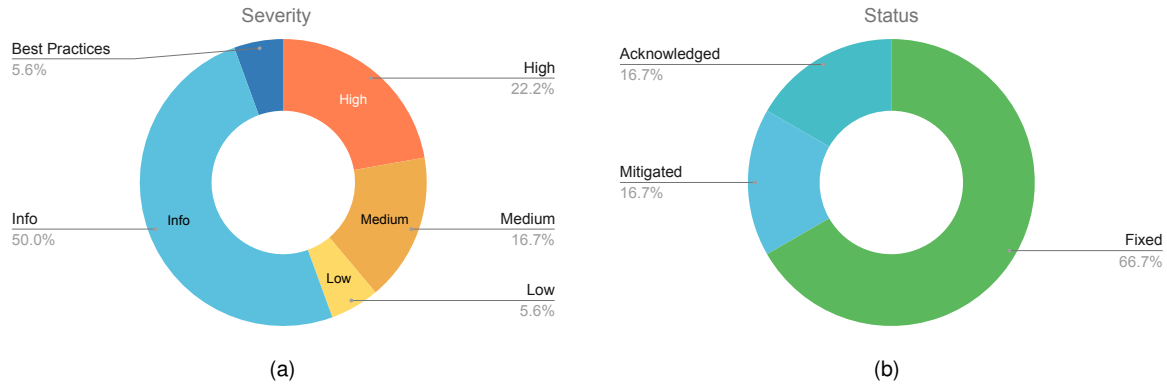

Fig. 1: Distribution of issues: Critical (0), **High** (4), **Medium** (3), **Low** (1), **Undetermined** (0), **Informational** (9), **Best Practices** (1). **Distribution of status: Fixed** (12), **Acknowledged** (3), **Mitigated** (3), **Unresolved** (0)

Summary of the Audit

Audit Type	Security Review
Initial Report	September 18, 2025
Final Report	October 04, 2025
Initial Commit	05ce1f8e4bea8508719e7f195be55b8bf4dfe36d
Final Commit	6111f9efe576544efbb93c8a1dc08854b698a325
Documentation Assessment	High
Test Suite Assessment	High

2 Audited Files

	Contract	LoC	Comments	Ratio	Blank	Total
1	DeveloperPool.sol	64	71	110%	26	161
2	InspectorPool.sol	64	70	109%	25	159
3	RegeneratorRules.sol	274	262	95%	108	638
4	ActivistPool.sol	64	70	109%	26	160
5	InspectionRules.sol	260	224	86%	103	587
6	RegenerationCredit.sol	56	103	183%	33	192
7	DeveloperRules.sol	183	276	150%	99	558
8	ValidationRules.sol	119	113	95%	57	289
9	CommunityRules.sol	192	190	99%	71	453
10	RegenerationIndexRules.sol	113	47	41%	23	183
11	InspectorRules.sol	148	208	140%	75	431
12	ContributorPool.sol	64	70	109%	26	160
13	ActivistRules.sol	122	165	135%	64	351
14	InvitationRules.sol	106	92	86%	39	237
15	ResearcherRules.sol	233	239	102%	104	576
16	RegeneratorPool.sol	65	71	109%	26	162
17	ResearcherPool.sol	64	70	109%	25	159
18	RegenerationCreditImpact.sol	57	89	156%	27	173
19	SupporterRules.sol	121	110	90%	54	285
20	VoteRules.sol	75	63	84%	25	163
21	ContributorRules.sol	202	275	136%	99	576
22	PoolTypes.sol	6	7	116%	1	14
23	SupporterTypes.sol	18	21	116%	2	41
24	ContributorTypes.sol	35	31	88%	5	71
25	IndexTypes.sol	14	13	92%	3	30
26	ResearcherTypes.sol	56	62	110%	7	125
27	ActivistTypes.sol	13	15	115%	2	30
28	ValidationTypes.sol	11	4	36%	1	16
29	DeveloperTypes.sol	36	26	72%	6	68
30	CommunityTypes.sol	34	60	176%	4	98
31	InspectorTypes.sol	21	22	104%	3	46
32	RegeneratorTypes.sol	27	30	111%	4	61
33	InspectionTypes.sol	33	23	69%	3	59
34	Callable.sol	21	38	181%	10	69
35	Poolable.sol	69	97	140%	32	198
36	Invitable.sol	13	27	207%	4	44
37	Blockable.sol	46	82	178%	24	152
38	IActivistPool.sol	8	30	375%	5	43
39	IInspectionRules.sol	6	20	333%	3	29
40	IContributorRules.sol	11	40	363%	8	59
41	IInspectorRules.sol	15	61	406%	12	88
42	IRegenerationIndexRules.sol	4	14	350%	1	19
43	IValidationRules.sol	10	19	190%	4	33
44	ICommunityRules.sol	20	79	395%	15	114
45	IDeveloperPool.sol	9	35	388%	6	50
46	IDeveloperRules.sol	11	40	363%	8	59
47	IInspectorPool.sol	8	30	375%	5	43
48	IResearcherRules.sol	12	45	375%	9	66
49	IRegeneratorPool.sol	9	36	400%	6	51
50	IVoteRules.sol	4	11	275%	1	16
51	IRegenerationCredit.sol	12	18	150%	9	39
52	IResearcherPool.sol	9	35	388%	6	50
53	IContributorPool.sol	9	35	388%	6	50
54	IRegeneratorRules.sol	19	56	294%	12	87
55	IActivistRules.sol	11	43	390%	8	62
	Total	**3286**	**4083**	**124.3%**	**1370**	**8733**

3 Summary of Issues

	Finding	Severity	Update
1	Fraud Reporting Mechanism Fails to Protect Protocol Integrity	High	Fixed
2	Inspectors Are Unfairly Penalized When Detecting Invalid Regenerators	High	Fixed
3	Inspectors Can Inflate Ecological Value to Profit from Token Sales	High	Fixed
4	Insufficient validation of coordinates can lead to double registration of a regeneration area	High	Mitigated
5	Any type of user can accept invitation from denied users	Medium	Fixed
6	Risk of Fake Delations Undermining User Reputation System	Medium	Fixed
7	Unexecuted Inspections may Block Regenerator Rewards	Medium	Fixed
8	The average level decreases when denied users are included in the vote power computation	Low	Fixed
9	Accumulation of Residual Tokens in Pools	Info	Acknowledged
10	Imbalanced Token Distribution in Inspector Pool Reduces Incentives	Info	Mitigated
11	Inspectors are deactivated in their three give-ups	Info	Fixed
12	Lack of Incentives for Inspection Invalidation	Info	Fixed
13	Permanent Pool Misconfiguration Risk After Ownership Renouncement	Info	Mitigated
14	Unallocated tokens in `Eras` and `Epochs` will be completely locked in the contract	Info	Acknowledged
15	Unclaimed Rewards Across Multiple Eras May Become Unrecoverable	Info	Acknowledged
16	_canVoteRules function returns true when totalUsers is zero	Info	Fixed
17	`canInvite` function returns true when `totalUsers` is zero	Info	Fixed
18	Event in `_denyUseris` never emitted	Best Practices	Fixed

4 Protocol Overview

The Regeneration Credit Protocol is a modular set of smart contracts designed to issue, distribute, and retire a fixed supply of Regeneration Credit (RC) tokens as a reward for ecosystem regeneration. The system coordinates multiple user roles `Regenerators`, `Inspectors`, `Activists`, `Researchers`, `Contributors`, `Developers`, `Supporters` through a combination of `Rules contracts`, `Pool contracts`, and `Shared modules`.

In the Regeneration Credit Protocol, each class of participant (actor) is managed by a **Rules contract** that enforces role-specific constraints (e.g., registration, validation, eligibility). These Rules contracts are tightly coupled with corresponding **Pool contracts**, which handle reward distribution based on **levels, inspections, or contributions**. The Pools themselves inherit from a shared`Poolable` abstraction that applies era/epoch logic and halving schedules. Finally, every actor (Supporter, Regenerator, Inspector, Activist, Researcher, Contributor, Developer) interacts with its own Rules contract as the primary entry point. The Rules contracts then forward state updates to the Pool contracts for reward accrual, while also interacting with `CommunityRules` for global registration, invitations, and denial mechanics.

This three-layer pattern is:

- **Actor ↔ Rules Contract**: users register, submit work (contributions, inspections, offsets), or withdraw rewards.
- **Rules ↔ Pool Contract**: Rules contracts add/remove pool levels or trigger withdrawals on behalf of actors.
- **Rules ↔ CommunityRules**: ensures proportionality, invitation validity, penalties, and role assignment system-wide.

4.1 Actors and Their Roles

- **Supporters:** They buy RC tokens and burn them to generate Impact Certificates.
- **Regenerators:** They are the primary value creators who register and regenerate land areas.
- **Inspectors:** Validate regenerators' projects with evidence (photos, counts, reports)
- **Activists:** Drive protocol growth by inviting new Regenerators and Inspectors.
- **Researchers:** Provide scientific methodologies and impact metrics
- **Contributors:** Provide general contributions (education, documentation, awareness campaigns).
- **Developers:** Provide technical infrastructure.

4.2 Structure of the Contracts and Interaction with Users

Figure 2 presents a flow diagram illustrating the core processes of the protocol and the interaction with the respective user roles.

Initialization: The protocol starts with the *owner*, who adds the pool contracts and transfers the predefined fixed amount of tokens to each pool.

User Onboarding: All users, with the exception of *Supporters*, must be invited to join the protocol. Invited users accept the invitation by calling the respective contract function to register themselves. For example, a *Regenerator* calls the `addRegenerator` function and subsequently the `requestInspection` function to register and request an inspection.

Inspection Workflow: Once an inspection request is opened by a Regenerator:

a. An *Inspector* must call the `acceptInspection` function to be assigned to the inspection.

b. After visiting the area and completing the inspection, the Inspector calls the `realizeInspection` function to register the ecological data (number of trees and biodiversity).

c. Each Inspector may inspect a given Regenerator only once.

Inspection Validation: In cases where an inspection is suspected to be invalid:

a. Users with voting rights call the `addInspectionValidation` function to challenge the inspection.

b. An inspection is invalidated only when the required minimum number of votes is reached.

c. Once invalidated, the Inspector who submitted the inspection is penalized.

Penalties and Restrictions: If an Inspector accumulates the maximum number of allowed penalties, their status is set to `denied`. A denied user cannot perform any further operations in the system, such as:

i) Accepting inspections,

ii) Withdrawing rewards,

iii) Inviting new users,

iv) Voting.

Similar penalty mechanisms apply to other user roles, except for *Supporters*, who are exempted from such restrictions.

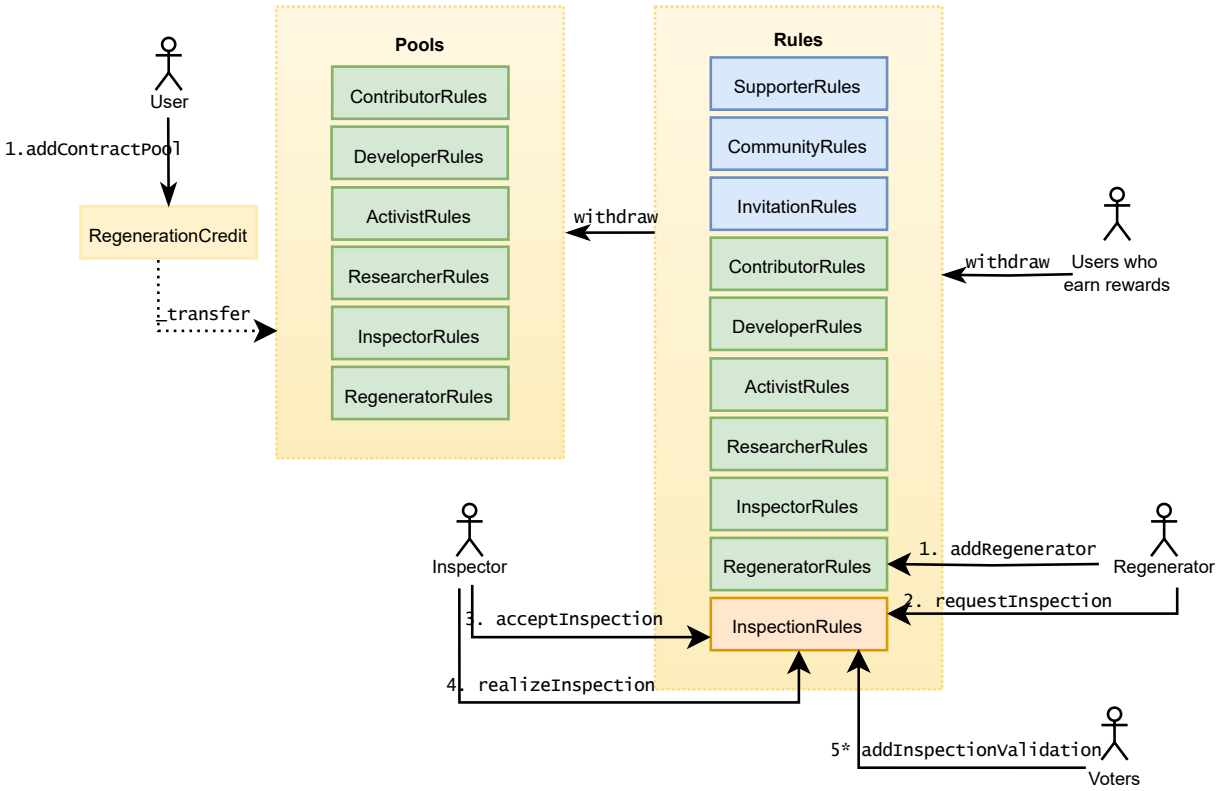

Fig. 2: Flow diagram for the core calls

5 Risk Rating Methodology

The risk rating methodology used by Nethermind Security follows the principles established by the OWASP Foundation. The severity of each finding is determined by two factors: **Likelihood** and **Impact**.

Likelihood measures how likely the finding is to be uncovered and exploited by an attacker. This factor will be one of the following values:

a) **High**: The issue is trivial to exploit and has no specific conditions that need to be met;

b) **Medium**: The issue is moderately complex and may have some conditions that need to be met;

c) **Low**: The issue is very complex and requires very specific conditions to be met.

When defining the likelihood of a finding, other factors are also considered. These can include but are not limited to motive, opportunity, exploit accessibility, ease of discovery, and ease of exploit.

Impact is a measure of the damage that may be caused if an attacker exploits the finding. This factor will be one of the following values:

a) **High**: The issue can cause significant damage, such as loss of funds or the protocol entering an unrecoverable state;

b) **Medium**: The issue can cause moderate damage, such as impacts that only affect a small group of users or only a particular part of the protocol;

c) **Low**: The issue can cause little to no damage, such as bugs that are easily recoverable or cause unexpected interactions that cause minor inconveniences.

When defining the impact of a finding, other factors are also considered. These can include but are not limited to Data/state integrity, loss of availability, financial loss, and reputation damage. After defining the likelihood and impact of an issue, the severity can be determined according to the table below.

		Severity Risk		
Impact	**High**	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Info/Best Practices	Low	Medium
	Undetermined	Undetermined	Undetermined	Undetermined
		Low	**Medium**	**High**
		Likelihood		

To address issues that do not fit a High/Medium/Low severity, Nethermind Security also uses three more finding severities: **Informational**, **Best Practices**, and **Undetermined**.

a) **Informational** findings do not pose any risk to the application, but they carry some information that the audit team intends to pass to the client formally;

b) **Best Practice** findings are used when some piece of code does not conform with smart contract development best practices;

c) **Undetermined** findings are used when we cannot predict the impact or likelihood of the issue.

6 Issues

6.1 [High] Fraud Reporting Mechanism Fails to Protect Protocol Integrity

File(s): `RegeneratorRules.sol`, `CommunityRules.sol`

Description: The system currently allows a user to become a regenerator only via invitation from an activist. There is no implementation to avoid that an area is used by another regenerator. While this mechanism restricts entry, it introduces vulnerabilities:

a - *Self-Invitation Abuse*: A user with an activist account can invite their own secondary accounts to join as regenerators, enabling them to register the same area multiple times.

b - *Area Misappropriation*: An activist can invite another account as regenerator but assign an area that belongs to a different regenerator, effectively diverting rewards to themselves across both activist and regenerator roles.

This can lead to duplicated or fraudulent area registrations. To mitigate the issue, the protocol introduces a denunciation mechanism that allows active users to flag irregularities.

However, the system does not effectively incentivize users to do so. For example, if an inspector invests time and money to visit an area and realizes they have already inspected it before for another regenerator, they may choose not to report the irregularity.

Recommendation(s): Reevaluate the denunciation mechanism to better incentivize users to report fraud and preserve the integrity of the protocol.

Status: Fixed

Update from the client: PR #664. The problem pointed at this issue is that the system does not block or calculate the regenerators area on chain. The reason for that is that it is complicated to put on chain real world assets without a centralized solution, in this case an area being regenerated. It is not possible, as far as I know, to solve this problem only with the on chain registration function. So the solution is the validation mechanism, which was updated to solve this problem.

The attack pointed here is the "malicious activist" attack, where a user with this role tries to obtain rewards for fake or duplicated regenerators area. First, it is not that the user will make an instant profit out of it, to be successfull new regenerators and inspectors need to conclude 3 valid inspections each one. And to do this without being discovered will be a challenge. A basic protection mechanism is the on chain reputation of the user, so the attacker will risk loose his account, what can return rewards/profit if acting honestly, when registering fake or duplicated areas.

One implementation to reduce the problem was to set activist as a non voter user, to avoid conflict of interest and to reduce this user type power. So now the voting rights and the inspections operations are separated. Activist, regenerators and inspectors are the operation base. And developers, researches and contributors are the voting layer, users that will have the validation as their function to the system. Another implementation was the reduction to only 5 allowed inviterPenalties for the Activist. So now, if 5 invited users gets denied, the activist is blocked from sending new invitations. So the risk is higher for the attacker.

```
constructor(
  uint8 inspectorProportionality,
  uint8 activistProportionality,
  uint8 researcherProportionality,
  uint8 developerProportionality,
  uint8 contributorProportionality
) {
  // ..
  userTypeSettings[CommunityTypes.UserType.ACTIVIST] = CommunityTypes.UserTypeSetting(
    activistProportionality,
    false,
    true,
    VOTER_INVITATION_DELAY_BLOCKS,
    false
  );
```

Another implementation was the ValidationPool, a token reward mechanism for hunters and voters. This implementation added a mechanism were hunters, or the users that start the voting for another user, receives one level of this validation pool if the user gets denied. The intention is to be a small pool, to reward a bonus or a tip for this service. So users can open voting, make "campaign" to invalidate a particular malicious user and get rewarded for that.

```
/**
 * @notice Allows users to attempt to vote to invalidate an user.
 * @dev Votes to invalidate users with unwanted behavior.
 *
 * Requirements:
 * - The caller must be a registered voter user (verified by VoteRules).
 * - Caller level must be above average (verified by VoteRules.canVote implicitly).
 * - Caller must have waited `timeBetweenVotes` since their last vote.
 * - Caller must vote only once per user per era.
```

```
   * - The target user must be registered and not already denied.
   * - If the target user is a Regenerator, they must have fewer than 4 completed inspections to be eligible for
    ↪ invalidation.
   *
   * @param userAddress Invalidation user address.
   * @param justification Invalidation justification (Max characters).
   */
  function addUserValidation(address userAddress, string memory justification) external nonReentrant {
    require(bytes(justification).length <= MAX_JUSTIFICATION_LENGTH, "Max characters");
    require(voteRules.canVote(msg.sender), "Not a voter");
    require(!communityRules.userTypeIs(CommunityTypes.UserType.UNDEFINED, userAddress), "User not registered");
    require(
      !communityRules.userTypeIs(CommunityTypes.UserType.SUPPORTER, userAddress),
      "Supporter validation not allowed"
    );
    require(!communityRules.isDenied(userAddress), "User already denied");
    require(_canBeValidated(userAddress), "Regenerator has reached validation immunity");

    uint256 currentEra = _userCurrentEra(userAddress);

    require(!validatorUsersValidations[msg.sender][userAddress][currentEra], "Already voted");
    require(waitedTimeBetweenVotes(msg.sender), "Wait timeBetweenVotes");

    if (hunterPools[msg.sender].currentEra == 0) {
      hunterPools[msg.sender].currentEra = validationPool.currentContractEra();
    }

    if (userValidations[userAddress][currentEra] == 0) {
      hunterVoter[userAddress][currentEra] = msg.sender;
    }

    validatorUsersValidations[msg.sender][userAddress][currentEra] = true;
    validatorLastVoteAt[msg.sender] = block.number;
    userValidations[userAddress][currentEra]++;
    validationPoints[msg.sender]++;

    uint256 validationsCount = userValidations[userAddress][currentEra];
    uint256 _votesToInvalidate = votesToInvalidate();

    if (validationsCount >= _votesToInvalidate) {
      _denyUser(userAddress);
      address hunter = hunterVoter[userAddress][currentEra];
      totalValidationLevels[hunter]++;
      validationPool.addLevel(hunter, userAddress);
    }

    emit UserValidation(msg.sender, userAddress, justification, currentEra);
  }


  /**
   * @dev Allows a validator to initiate a withdrawal of Regeneration Credits
   * for the malicious/fake users hunt service. Rewards will be based on their hunting level and current era.
   * @notice Validators can claim tokens for their hunt and investigation service.
   */
  function withdraw() external nonReentrant {
    // Only registered voters can call this function.
    require(communityRules.isVoter(msg.sender), "Pool only to voters");
    require(validatorLastVoteAt[msg.sender] > 0, "Not eligible to withdraw");

    Pool storage hunterPool = hunterPools[msg.sender];
    uint256 currentEra = hunterPool.currentEra;

    // Check if the validator is eligible to withdraw for the current era through DeveloperPool.
    require(validationPool.canWithdraw(currentEra), "Not eligible to withdraw for this era");

    // Increment the validator's era in their local pool data.
    hunterPool.currentEra++;

    // Call the DeveloperPool contract to perform the actual token withdrawal.
    validationPool.withdraw(msg.sender, currentEra);
  }
```

The other voters, they do have the incentive to vote to keep the system clean and increase their past and future rewards by having a community without the fraudlent actors. But to improve their incentive, a validationPoints mechanism was added to all vote functions, and

with 50 validationPoints, the user can exchange his points for one validationPool level and receive a token tip for this service.

```
/// @notice Validation points required for one pool level.
uint256 public constant POINTS_PER_LEVEL = 50;

/// @notice Tracks the accumulated, unspent validation points for each voter.
mapping(address => uint256) public validationPoints;

/// @notice Tracks the total number of validation levels a user has ever earned.
mapping(address => uint256) public totalValidationLevels;

/**
 * @notice Grants a single validation point to a voter for a voting action.
 * @dev This is a function intended to be called by the resources contracts after a validation vote.
 * @param voter The address of the voter who is earning the point.
 */
function addValidationPoint(address voter) external mustBeAllowedCaller {
  validationPoints[voter]++;
}

/**
 * @notice Allows a voter to exchange their accumulated validation points for a single level.
 * @dev This function implements a fixed exchange rate where a voter can trade a specific
 * amount of points (POINTS_PER_LEVEL) for one level, which contributes to their
 * standing and potential rewards in the Validation Pool.
 *
 * Requirements:
 * - The caller (`msg.sender`) must be a registered voter (e.g., Researcher, Developer, Contributor).
 * - The caller must have accumulated at least `POINTS_PER_LEVEL` points to be eligible for the exchange.
 */
function exchangePointsForLevel() external nonReentrant {
  require(communityRules.isVoter(msg.sender), "Pool only to voters");

  uint256 userPoints = validationPoints[msg.sender];
  require(userPoints >= POINTS_PER_LEVEL, "Not enough points");

  if (hunterPools[msg.sender].currentEra == 0) {
    hunterPools[msg.sender].currentEra = validationPool.currentContractEra();
  }

  validationPoints[msg.sender] = userPoints - POINTS_PER_LEVEL;
  totalValidationLevels[msg.sender]++;

  validationPool.addPointsLevel(msg.sender);
}
```

Also, some events had the indexed tag added to facilitate lookup.

6.2 [High] Inspectors Are Unfairly Penalized When Detecting Invalid Regenerators

File(s): `CommunityRules.sol`, `InspectionRules.sol`

Description: The protocol provides mechanisms to invalidate inspections and to report other users or resources. However, when an inspector accepts an inspection and later realizes that the area belongs to a regenerator they have already audited (which is not allowed), the current design leaves the inspector with only two unfavorable outcomes:

1. **Abandon the inspection** — The inspector does not submit, but is penalized for giving up, which may eventually lead to exclusion from the system;
2. **Submit with denunciation** — The inspector calls the `InspectionRules.realizeInspection` function to complete the inspection. Report the regenerator calling `CommunityRules.addDelation` indicating the regenerator's address. The inspection is then subject to voting and invalidation, which also penalizes the inspector when the inspection is invalidated. If the inspector does not report the regenerator, they also may be penalized if users invalidate the inspection.

Inspectors are punished in both cases, even when acting honestly.

Recommendation(s): Revise the invalidation and denunciation logic to ensure that inspectors acting correctly are not penalized.

Status: Fixed

Update from the client: Fixed at PR #662. This problem was solved by introducing a check to only count as a realized inspection and increment the ecological counters if the regeneration score is positive. Also, was added a check to only add pool level and increment inspections count to both regenerators and inspectors if score is not zero.

We will encourage inspectors to realize the inspection passing 0 as value and with his description/delation of the problem faced on the report. So with this update, 0 score inspection will not afect the token ecological backing, will not increase level and inspections count, will not increase the inspector giveUp count and may not be penalized if well justified at the report.

This way, the inspector don't even need to create a delation, the the zero inspection with the justification for the invalid regenerator. No giveUps, no penalty, if validators agree with the justification.

Impact update:

```
/**
 * @dev Allow a inspector realize a inspection and mark as INSPECTED.
 * @notice Inspectors must evaluate the amount of trees and species of the regeneration area.
 * How many trees, palm trees and other plants over 1m high and 3cm in diameter there is in the regenerating area?
 ↪ Justify your answer in the report.
 * How many different species of those plants/trees were found? Each different species is equivalent to one unity and
 ↪ only trees and plants managed or planted by the regenerator should be counted. Justify your answer in the report.
 * Zero score means invalid inspection.
 * NOTE: If the inspector finds something suspicous about the inspected regenerator, such as invalid area, suspicious
 ↪ of fake account, or if the Regenerator is not
 * findable, inspectors are encourage to realize passing 0 as values with his justification at the report to avoid
 ↪ being penalized.
 * @param inspectionId The id of the inspection to be realized.
 * @param proofPhotos The string of a photo with the regenerator or the string of a document with the proofPhoto with
 ↪ the regenerator and other area photos.
 * @param justificationReport The justification and report of the result found.
 * @param treesResult The number of trees, palm trees and other plants over 1m high and 3cm in diameter found in the
 ↪ regeneration area. Only plants managed or planted by the regenerator must be counted.
 * @param biodiversityResult The number of different species of trees, palm trees and other plants over 1m high and 3cm
 ↪ in diameter found in the regeneration area. Only plants managed or planted by the regenerator must be counted.
 */
function realizeInspection(
  uint64 inspectionId,
  string memory proofPhotos,
  string memory justificationReport,
  uint32 treesResult,
  uint32 biodiversityResult
) external nonReentrant {
  // ...

  // Only count inspections that have a positive impact towards the global metrics.
  if (inspection.regenerationScore > 0) {
    impactPerEra[inspection.inspectedAtEra].trees += treesResult;
    impactPerEra[inspection.inspectedAtEra].biodiversity += biodiversityResult;
    realizedInspectionsCount++;
    impactPerEra[era].realizedInspections++;
  }
// ...
}
```

To solve the problem that inspections not carried out were still counting to the totalInspections, a new logic to only count as realized inspection was introduced. Now, a zero score inspection will not be counted as an inspection or a level for users.

This was implemented at the afterRealizeFunction for both inspectors and regenerators:

```
  function afterRealizeInspection(
    address addr,
    uint32 score,
    uint64 inspectionId
  ) external mustBeAllowedCaller mustBeContractCall(inspectionRulesAddress) nonReentrant returns (uint256) {
    require(score <= MAX_SCORE, "Maximum score");
    require(!processedInspections[inspectionId], "Inspection results already submitted");
  // ...
    if (score > 0) {
      totalInspections = _incrementInspections(addr);
    } else {
      totalInspections = regenerators[addr].totalInspections;
    }
  // ...
  }
```

6.3 [High] Inspectors Can Inflate Ecological Value to Profit from Token Sales

File(s): RegenerationCreditImpact.sol

Description: The functions `treesPerToken`, `carbonPerToken`, and `biodiversityPerToken` compute the per-token ecological impact (trees, carbon, and biodiversity, respectively). Each of these functions ultimately depends on the number of trees recorded in the system:

```
function treesPerToken() external view returns (uint256) {
  uint256 effectiveSupply = _getEffectiveSupply();
  if (effectiveSupply == 0) return 0;

  return (totalTreesImpact() * PRECISION_FACTOR) / effectiveSupply;
}

function carbonPerToken() external view returns (uint256) {
  uint256 effectiveSupply = _getEffectiveSupply();
  if (effectiveSupply == 0) return 0;

  return (totalCarbonImpact() * PRECISION_FACTOR) / effectiveSupply;
}
```

The values of `totalTreesImpact` and `totalCarbonImpact` are derived from `inspectionsTreesImpact`, which accumulates the number of trees reported across all inspections:

```
function totalTreesImpact() public view returns (uint256) {
  if (inspectionRules.realizedInspectionsCount() == 0) return 0;

  return
    (inspectionRules.inspectionsTreesImpact() * regeneratorRules.totalImpactRegenerators()) /
    inspectionRules.realizedInspectionsCount();
}

function totalCarbonImpact() public view returns (uint256) {
  return totalTreesImpact() * CARBON_PER_TREE;
}
```

Each inspection contributes to inspectionsTreesImpact, capped at four trees per square meter and a maximum inspected area of 500,000 sqm. If an inspection is later invalidated by voters, both inspectionsTreesImpact and realizedInspectionsCount are decremented accordingly.

The problem arises when a malicious user operates multiple accounts as inspectors or independent inspectors observe that the Regeneration Credit (RC) value is connected to the number of trees. Since inspectors can take on new inspections roughly every 24 hours, they can repeatedly submit manipulated inspections to artificially inflate the ecological value of the RC token and then sell tokens earned in previous eras (e.g., Era n-1, n-2, etc.).

This vulnerability allows attackers to artificially manipulate the ecological value per RC token, enabling unfair profit extraction and undermining the credibility and stability of the system.

Recommendation(s): Review the design of the per-token ecological impact calculation to ensure inspectors cannot manipulate token value. One possible mitigation is to base the computation solely on validated data from the previous era.

Status: Fixed

Update from the client: A struct impactEra and a mapping from era to impactEra was introduced. Now the impact calculation is realized by era. A setEraImpact function was added at the request inspection function to add the past era impact to the global metrics. So now the impact will be increased at the first requestInspection of each era, summing only the past impact to avoid the manipulation issue. Fixed at PR #663.

```
/**
 * @notice Tracks the inspection impact of an Era.
 * @dev This struct is used to register the impact of all inspection of an Era.
 * @param trees Trees count.
 * @param biodiversity Biodiversity count.
 * @param realizedInspections Era realizedInspections count.
 */
struct EraImpact {
  uint256 trees;
  uint256 biodiversity;
  uint256 realizedInspections;
}


  function realizeInspection(
    uint64 inspectionId,
    string memory proofPhotos,
    string memory justificationReport,
```

```
    uint32 treesResult,
    uint32 biodiversityResult
  ) external nonReentrant {
    // ..
    if (inspection.regenerationScore > 0) {
      uint256 era = inspection.inspectedAtEra;
      impactPerEra[era].trees += treesResult;
      impactPerEra[era].biodiversity += biodiversityResult;
      impactPerEra[era].realizedInspections++;
    }
    // ..
}

function requestInspection() external nonReentrant {
  // ..

  // Update era impact.
  _setEraImpact();
  }

function _setEraImpact() private {
    uint256 nextEraToSet = lastSettledEra + 1;

    if (nextEraToSet < regeneratorRules.poolCurrentEra()) {
      EraImpact storage eraImpact = impactPerEra[nextEraToSet];

      inspectionsTreesImpact += eraImpact.trees;
      inspectionsBiodiversityImpact += eraImpact.biodiversity;
      realizedInspectionsCount += eraImpact.realizedInspections;
      totalImpactRegenerators += regeneratorRules.newCertificationRegenerators(nextEraToSet);
      // Update the lastSetlledEra to the era just settled.
      lastSettledEra = nextEraToSet;
    }
 }
```

To avoid variables conflict, the logic to calculate the impact regenerators, users that are on the certification process was changed and now a mapping registers how many new impact regenerators started at each era.

```
  /// @notice The number of regenerators that have started the certification process on each era,
  /// and have reached the minimum of one inspection.
  mapping(uint256 => uint256) public newCertificationRegenerators;

  function decrementInspections(address addr) external
    // ..
    if (totalInspections == 1) {
      uint256 era = poolCurrentEra();
      if (newCertificationRegenerators[era] > 0) {
          newCertificationRegenerators[era]--;
      }
      impactRegenerators[addr] = false;
    }
    // ..
  }

  function _incrementInspections(address addr) private returns (uint256) {
    // ..
    if (!impactRegenerators[addr]) {
      impactRegenerators[addr] = true;
      uint256 era = poolCurrentEra();
      newCertificationRegenerators[era]++;
    }
    // ..
  }
```

6.4 [High] Insufficient validation of coordinates can lead to double registration of a regeneration area

File(s): RenegeratorRules.sol

Description: When adding a `regenerator` the `addRegeneratior` function validates the provided coordinates via the `_validateCoordinates` function presented below:

```
function _validateCoordinates(Coordinates[] calldata _coords) private pure {
   // Loop through each coordinate
   for (uint256 i = 0; i < _coords.length; i++) {
     // --- 1. Check for Duplicates ---
     // Compare the current coordinate with all subsequent coordinates
     for (uint256 j = i + 1; j < _coords.length; j++) {
       // We use keccak256 to compare the structs efficiently
       require(
         keccak256(abi.encode(_coords[i])) != keccak256(abi.encode(_coords[j])), //@audit can represent same area ie
          ↪  13.23 is the same as 13.230000.
         "Duplicate coordinates are not allowed"
       );
     }

     // ....
     // ....
   }
 }

```

In validating the coordinates structs the function does `keccak256(abi.encode(_coords[i])) != keccak256(abi.encode(_coords[j]))` operation. Now given this is a `keccak256` hash operation then any variability in the coordinates data will not match and the check will pass. For example a latitude of `13.2` will produce a different hash from `13.2000` and in coordinates the value `13.2000` is basically the same area but with a higher precision.

Recommendation(s): Consider improving validation of provided coordinates. For example the difference between coordinates 12.3456 and 12.3457 is only 11 meters and this might represent the same regeneration area.

Status: Mitigated

Update from the client: Mitigated at PR #659. The problem about on-chain coordinates storage is that it is technically complicated to compare these points with a solidity function to, for example, block duplicate registration or polygon crossings areas. We could develop a way to check duplicate coordinates points, but if the user walks 1m on any direction than it is already a different point that could be stored instead. So the system needs to rely on the community to perform the cleaning actions to check the area. The challenge for a governance system is that it requires user interaction and participation. Hunters will need to review every single registration and voters will need to proactively vote for maintenance of the system.

The question about this function problem, is a mathematical problem that allows 13.2 and 13.2000 to be registered as if it was different points. The function was updated to solve this problem and block now the regenerator to add the same coordinates at registration, solving the mathematical problem. The issue abount the duplicated coordinates or crossing area polygons still exists and perhaps is unsolvable in solidity. We could have developed a way to store all coordinate points and then check at registration. It would be necessary 2 fors going from 3-10 coordinates to make this check, adding a good amount of gas to the function. Since even with this check it would still need the validation process, we decided to keep the function cleaner without the seenPoints logic. Here is the new validateCoordinates function:

```
 function _validateCoordinates(Coordinates[] calldata _coords) private pure {
   uint256 len = _coords.length;
   int256 precision = 10 ** 6;

   for (uint256 i = 0; i < len; i++) {
     // --- 1. Convert and Validate Range ---
     int256 lat_i = _stringCoordToInt(_coords[i].latitude);
     int256 lon_i = _stringCoordToInt(_coords[i].longitude);

     require(lat_i >= -90 * precision && lat_i <= 90 * precision, "Invalid latitude");
     require(lon_i >= -180 * precision && lon_i <= 180 * precision, "Invalid longitude");

     // --- 2. Check for Duplicates ---
     for (uint256 j = i + 1; j < len; j++) {
       int256 lat_j = _stringCoordToInt(_coords[j].latitude);
       int256 lon_j = _stringCoordToInt(_coords[j].longitude);

       require(lat_i != lat_j || lon_i != lon_j, "Duplicate coordinates are not allowed");
     }
   }
 }
```

6.5 [Medium] Any type of user can accept invitation from denied users

File(s): ActivistRules.sol, DeveloperRoles.sol, ContributorRules.sol, RegeneratorRules.sol, ResearcherRules.sol,

Description: The system defines different user types: activist, researcher, developer, and contributor. Each of which requires an invitation to join the community. Only users without the denied status are permitted to send invitations.

However, after an invitation is sent and before the invitee completes their registration, the inviter may subsequently be blocked from performing any actions, including sending new invitations. Despite this, there is no verification at the time of registration to ensure that the inviter remains eligible.

This allows invitees to successfully register using invitations from inviters who have since been blocked, undermining the restriction mechanism and enabling indirect circumvention of access controls. This issue impacts all the rule contracts implemented for the mentioned user type above.

Recommendation: Validate the inviter's status at the time of invitee registration to ensure that invitations from blocked users cannot be used. The `addUser` function in the `CommunityRules` contract is invoked by all user registration flows. For better modularity and to reduce the risk of errors, the necessary validation should be implemented directly within this function.

```
function addUser(address addr, CommunityTypes.UserType userType) external mustBeAllowedCaller {
  require(addr != address(0), "User address cannot be zero");
  require(users[addr] == CommunityTypes.UserType.UNDEFINED, "User already exists");
  require(userType != CommunityTypes.UserType.UNDEFINED, "Invalid user type");
  require(_registrationProportionalityAllowed(userType), "Proportionality invalid");
  require(_invitedTypeOnRegister(addr, userType), "Invalid invitation");
  // @audit insert the checking if the inviter is not denied

  users[addr] = userType;
  usersCount++;
  userTypesCount[userType]++;
  userTypesTotalCount[userType]++;

  emit UserRegistered(addr, userType);
}
```

Status: Fixed

Update from the client: A require check was added to the addUser function at CommunityRules.sol to block registration of invited wallets by a denied inviter. Now, if a inviter gets denied, all invitations previously made from that user are not valid anymore.

To increase security, besides the isDenied check, another require was added at addUser function to block registration from inviters that have exceeded the max inviter penalties. It is the same issue, just a different penalty that should also block invitation. Fixed at PR #658

```
  function addUser(address addr, CommunityTypes.UserType userType) external mustBeAllowedCaller {
    // ..
      require(!isDenied(invitations[addr].inviter), "Inviter denied"); // Inviter cannot be denied
      require(inviterPenalties[invitations[addr].inviter] < MAX_INVITER_PENALTIES, "Inviter with too many penalties");
      ↪  // Inviter cannot exceed penalties
    // ..
  }
```

6.6 [Medium] Risk of Fake Denunciation Undermining User Reputation System

File(s): CommunityRules.sol

Description: Currently, users can file a report alleging malicious behave on the protocol by other users. These denunciations are stored on the reported user's profile and are visible to everyone, effectively marking the profile as "dirty" when misbehavior is recorded.

The `addDelation` function can be called by any user, except for `supporters`. As shown, each denunciation is stored in the `delations` mapping. Currently, there is no function available to update or change the status of a denunciation; once created, it is automatically assumed to be valid.

```
function addDelation(address addr, string memory title, string memory testimony) external {
  require(
    bytes(title).length <= MAX_TITLE_LENGTH && bytes(testimony).length <= MAX_TESTIMONY_LENGTH,
    "Max characters reached"
  );
  require(hasWaitedRequiredTime(msg.sender), "Wait delay blocks");
  require(users[msg.sender] != CommunityTypes.UserType.UNDEFINED, "Caller must be registered");
  require(users[msg.sender] != CommunityTypes.UserType.SUPPORTER, "Not allowed to supporters");
  require(users[addr] != CommunityTypes.UserType.UNDEFINED, "User must be registered");
  require(addr != address(0), "Cannot delate zero address");
  require(addr != msg.sender, "Self-denunciation not allowed");

  lastDelationBlock[msg.sender] = block.number;

  delationsCount++;
  uint64 newDelationId = delationsCount;

  CommunityTypes.Delation memory newDelation = CommunityTypes.Delation(
    newDelationId,
    msg.sender,
    addr,
    title,
    testimony,
    block.number
  );

  delations[addr].push(newDelation);
  delationsById[newDelationId] = newDelation;

  emit DelationAdded(msg.sender, addr, newDelationId);
}
```

However, this mechanism introduces a critical risk: a malicious actor could deliberately create false denunciation to harm another user's reputation. Over time, as fake reports accumulate alongside legitimate ones, the overall value and credibility of this feature may diminish.

Recommendation(s): Implement a validation layer for denunciations. One possible approach is a voting mechanism, where other users (or a designated group) can validate or invalidate each denunciation. This would reduce the risk of abuse, increase the accuracy of reputation tracking, and help maintain trust in the system over time.

Status: Fixed

Update from the client: The problem was significantly reduced after the implementation of a check to only allow one delation per user. A user can still add a fake delation, riskying his own credibility, but only one. More than one delation per wallet per user is not allowed anymore.

The second mitigation action of this issue is a voting mechanism were users can vote true or false for a delation. With this logic, users can add a social proof and vote to unmask fake delations or support true delations. This way, even a fake delation with the intention to harm a users reputation can be largely voted as false, not compromising the user reputation.

With this change, the delation data structure was updated to split functions between mappings and leave the work to the front-end. One to store the ids and another with de relationship between delation id and its data. Fixed at PR #657

New mappings structure:

```
  /// @dev Index of delation IDs for each reported user.
  mapping(address => uint64[]) private _delationIdsForUser;

  /// @dev mapping: delationId => voterAddress => hasVoted (bool)
  mapping(uint64 => mapping(address => bool)) private _hasVotedOnDelation;

```

New voteOnDelation function:

```
  /**
   * @notice Allows users to vote (thumbs up/down) on an existing delation.
```

```
    * @dev This creates a social validation layer. Voters cannot be the informer or the reported user.
    * @param _delationId The ID of the delation to vote on.
    * @param _supportsDelation True for a "thumbs up" (agrees), false for "thumbs down" (disagrees).
    */
   function voteOnDelation(uint64 _delationId, bool _supportsDelation) external {
     // 1. Check if the delation exists by accessing it. It will revert if the ID is invalid.
     CommunityTypes.Delation storage delation = delationsById[_delationId];
     require(delation.id != 0, "Delation does not exist");

     // 2. Check if the voter is eligible.
     require(users[msg.sender] != CommunityTypes.UserType.UNDEFINED, "Caller must be registered");
     require(users[msg.sender] != CommunityTypes.UserType.SUPPORTER, "Not allowed to supporters");
     require(!isDenied(msg.sender), "User denied");

     // 3. The informer and the reported user cannot vote on their own delation.
     require(msg.sender != delation.informer, "Informer cannot vote");
     require(msg.sender != delation.reported, "Reported user cannot vote");

     // 4. Check to prevent double voting.
     require(!_hasVotedOnDelation[_delationId][msg.sender], "Already voted");

     // --- State Changes ---

     // Mark that this user has now voted.
     _hasVotedOnDelation[_delationId][msg.sender] = true;

     // Increment the appropriate counter.
     if (_supportsDelation) {
       delation.thumbsUp++;
     } else {
       delation.thumbsDown++;
     }

     emit DelationVoted(_delationId, msg.sender, _supportsDelation, delation.thumbsUp, delation.thumbsDown);
   }
```

6.7 [Medium] Unexecuted Inspections may Block Regenerator Rewards

File(s): InspectionRules.sol

Description: Inspectors accept inspection requests through the `acceptInspection` function, which updates the inspection status from `InspectionStatus.OPEN` to `InspectionStatus.ACCEPTED` and records the block number at which it was accepted:

```
function acceptInspection(uint64 inspectionId) external nonReentrant {//ok
 // ...
  require(inspection.status == InspectionStatus.OPEN, "Inspection must be OPEN");
 // ...

  inspection.status = InspectionStatus.ACCEPTED;
  inspection.acceptedAt = block.number;
  inspection.inspector = msg.sender;

  // ...
}
```

When the assigned inspector later calls realizeInspection to submit results, the contract checks that the inspection has not expired:

```
function realizeInspection(
  uint64 inspectionId,
  string memory proofPhotos,
  string memory justificationReport,
  uint32 treesResult,
  uint32 biodiversityResult
) external nonReentrant {
  // ...
  require(inspection.status == InspectionStatus.ACCEPTED, "Accept before");
  require(inspection.inspector == msg.sender, "Not your inspection");
  require(!(block.number > inspection.acceptedAt + blocksToExpireAcceptedInspection), "Inspection Expired");
  // ...
}
```

As presented in the code above, once the block.number exceeds acceptedAt + blocksToExpireAcceptedInspection, the inspector can no longer submit results.

While the protocol penalizes inspectors for failing to deliver, it also indirectly penalizes regenerators. If an inspection is created near the end of an era and the assigned inspector does not complete it in time, the regenerator may be unable to create a new inspection and have it realized before the era ends. As a result, the regenerator's rewards are delayed until the next era (around six months). In addition, each regenerator can request up to 11 inspections and earn rewards, and even those not carried out would still be counted within this limit.

Recommendation(s): This problem can be mitigated by allowing inspections with status `ACCEPTED` but already expired to be reassigned to another inspector. This can be achieved by adapting the `acceptInspection` function to include an additional condition that permits reassignment when an inspection has expired:

```
function acceptInspection(uint64 inspectionId) external nonReentrant {//ok
  require(communityRules.userTypeIs(CommunityTypes.UserType.INSPECTOR, msg.sender), "Only inspectors");
  require(inspectorRules.isInspectorValid(msg.sender), "Only 3 giveUps allowed");

  Inspection storage inspection = inspections[inspectionId];

  require(inspection.id >= 1, "Inspection do not exist");
  require(alreadyHaveInspectionAccepted(), "Already accepted");
  require(!inspectorInspected[msg.sender][inspection.regenerator], "Already inspected");
  // @audit Adapt the restriction to also allow inspections with status ACCEPTED that have already expired.
  require(inspection.status == InspectionStatus.OPEN, "Inspection must be OPEN");
  require(acceptInspectionDelayBlocksPassed(inspection), "Wait delay blocks");
  require(beforeAcceptHaveSecurityBlocksToVote(), "Wait until next era");
  // ...
}
```

Status: Fixed

Update from the client: A new function isInspectionExpired() was introduced to check if an existing inspection was expired. Then, the require inspection.STATUS == OPEN was incremented with the logic to check and allow other inspectors to accept the same inspection again, solving the reported issue.

The problem that inspections not carried out were still counting to the totalInspections was fixed at issue number #2. Fixed at PR #660

```
  /**
   * @dev Checks if a previously accepted inspection has expired.
   * @param inspection The inspection to check.
```

```
   * @return bool True if the inspection is expired, false otherwise.
   */
  function isInspectionExpired(Inspection memory inspection) public view returns (bool) {
    return inspection.acceptedAt > 0 && (block.number > inspection.acceptedAt + blocksToExpireAcceptedInspection);
  }

  function acceptInspection(uint64 inspectionId) external nonReentrant {
    // ..
    require(
      inspection.status == InspectionStatus.OPEN ||
        (inspection.status == InspectionStatus.ACCEPTED && isInspectionExpired(inspection)),
      "Inspection must be OPEN or EXPIRED"
    );
   // ..
  }
```

6.8 [Low] The average level decreases when denied users are included in the vote power computation

File(s): VoteRules.sol, ValidationRules.sol

Description The `canVote` function determines whether a user is eligible to vote based on a global user counter. However, the value in `userTypesTotalCount` includes both **active** and **denied** users. As a result, `totalUsers` is equal to `userTypesTotalCount`.

```
function canVote(address addr) public view returns (bool) {
    require(communityRules.isVoter(addr), "Not a voter user");

    CommunityTypes.UserType userType = communityRules.getUser(addr);
    uint256 totalUsers = communityRules.userTypesTotalCount(userType);

    // @audit The function should use only the number of active users to determine voting eligibility
    return _canVoteRules(_totalLevels(userType), totalUsers, _totalUserLevels(addr, userType));
}
```

Because the average level is calculated using this global counter, it becomes lower than intended, allowing users with fewer levels to pass the voting threshold.

```
function _canVoteRules(uint256 totalTypeLevels, uint256 totalUsers, uint256 userLevels) private pure returns (bool) {
    // @audit The average is skewed when the global counter includes denied users
    // Rule 2: Check if the user's level is strictly greater than the average
    return userLevels * totalUsers > totalTypeLevels;
}
```

In addition, there is an inconsistency with how voting power is computed. The `votesToInvalidate` function, which defines the threshold for invalidating an inspection, relies only on the count of **active users**:

```
function votesToInvalidate() public view returns (uint256) {
    uint256 voters = communityRules.votersCount();

    if (voters < VOTERS_THRESHOLD_LEVEL_1) {
      return VOTES_TO_INVALIDATE_LEVEL_1;
    }

    if (voters < VOTERS_THRESHOLD_LEVEL_2) {
      return VOTES_TO_INVALIDATE_LEVEL_2;
    }

    uint256 invalidationVotes = (voters * DYNAMIC_INVALIDATION_PERCENTAGE) / 100;
    return invalidationVotes + 1;
}
```

This discrepancy leads to inconsistent logic between **eligibility to vote** and **the number of votes required to invalidate**, although the practical severity of the issue is low since `votesToInvalidate` uses only active voters.

Recommendation Update the `canVote` logic to rely exclusively on **active users** when computing voting power. This ensures consistency with the `votesToInvalidate` function and prevents unintended lowering of the voting threshold.

Status: Fixed

Update from the client: A variable named 'activeLevelsCount' was added to each of the voters contract, adding a level when a resource is created and removing a level when the resource is invalidated. When the user is denied, all levels from that user is removed from the 'activeLevelsCount'. To solve this issue, it was necessary this implementation because if the function was simply updated to userTypesCount instead of userTypesTotalCount and resourcesCount instead of resourcesTotalCount it would lead to data incosistency because a user can be invalidated with past eras valid resources. For example a researcher that was denied at era 2 with valid researches at era 1. Now, the average to calculate if the user canVote, and the same was applied also to the canInvite function, is processed counting only valid users and their resources. A 'clean' average was implemented instead of the old 'dirty' average. Fixed at PR #653

New variable implementation:

```
  /// @notice The sum of all active levels from valid reports by non-denied developers.
  uint256 public totalActiveLevels;
```

The totalActiveLevel is being incremented at the functions that add level to the user. For example, the addContribution function for contributors:

```
  function addContribution(string memory description, string memory report) external nonReentrant {
    // ..
    contributionsCount++;
    contributionsTotalCount++;
```

```
    totalActiveLevels++;
    // ..
  }
```

And the same variable is being decremented by one level when a resource is invalidated or is removing all levels when the user is denied:

```
  function _invalidateContribution(Contribution memory contribution) private {
    contributionsCount--;
    contribution.valid = false;
    contribution.invalidatedAt = block.number;
    contributions[contribution.id] = contribution;
    contributors[contribution.user].pool.level -= RESOURCE_LEVEL;
    totalActiveLevels--;

    contributorPool.removePoolLevels(contribution.user, false);
  }
```

```
  function _denyContributor(address userAddress) private {
    if (communityRules.isDenied(userAddress)) return; // Already denied, nothing to do

    totalActiveLevels -= contributors[userAddress].pool.level;

    communityRules.setToDenied(userAddress);
    // ..
  }
```

And now the logic of the canVote and also canInvite are using this variable instead of the old dirty total count:

```
  function canSendInvite(address addr) public view returns (bool) {
    Developer memory developer = developers[addr];

    // Return false if the address is not a registered developer (id is 0).
    if (developer.id <= 0) return false;

    // Calls the inherited `canInvite` function from `Invitable` to calculate eligibility.
    // This depends on total reports count, total developer count, and the developer's pool level.
    return canInvite(totalActiveLevels, communityRules.userTypesCount(USER_TYPE), developer.pool.level);
  }
```

6.9 [Info] Accumulation of Residual Tokens in Pools

File(s): RegeneratorPool.sol, ActivistPool.sol, ResearcherPool.sol, InspectorPool.sol,

Description: After users withdraw from pools for a given era, small amounts of tokens often remain stuck. Over time, these residual tokens accumulate across multiple eras and epochs. In the current protocol design, any leftover tokens from past eras are effectively treated as lost and remain unusable.

Recommendation(s): Consider carrying over residual tokens to subsequent eras.

Status: Acknowledged

Update from the client: The design is to keep the leftover as lost and remain unusable.

6.10 [Info] Imbalanced Token Distribution in Inspector Pool Reduces Incentives

File(s): `RegeneratorPool.sol`, `InspectorPool.sol`

Description: According to the white paper and the code, fixed amounts of tokens are allocated to each user type's pool. While Researchers, Developers, Contributors, and Activists each receive 3.20% of the total supply, Regenerators and Inspectors receive significantly higher allocations of 60% and 14.40% respectively, as they are considered the core roles of the protocol.

- Regenerators register their areas in the system;
- Inspectors physically audit those areas, reporting biodiversity and tree counts;

However, this distribution introduces a structural imbalance. The number of Regenerators is expected to be substantially smaller than the number of Inspectors. Each Regenerator can request 11 inspections and one Inspector can only audit each Regenerator once. As a result:

- The Inspector pool will be spread across a much larger group of users, reducing individual rewards;
- Inspectors, who are crucial for validating Regenerators' submissions, may lack sufficient incentives to remain active;

Without multiple Inspectors available per Regenerator, the protocol risks failing to operate as intended.

Recommendation(s): Reevaluate the token distribution, particularly in the Inspector and Regenerator pools, to ensure adequate incentives that support sustainable participation and maintain protocol integrity.

Status: Mitigated

Update from the client: The reason the regenerator pool is much bigger is that they are the ecological foundation of the protocol. The onchain service of the inspector is much harder and costly then the simple requestInspection call for the regenerator. But the real service of the regenerators is on the ground, planting and managing the threes and the ecosystem of the registered area.

The act of realizing inspections is the foundation of the system, it will set the ecological base and growth speed. In a scenario with more inspections then inspectors interested on realizing then, the system growth will be limited by this operation. And at the scenario where more inspectors want participate, this will drive competition for accepting the inspections. The natural process is that overtime this balance will change. More inspectors, means more competition, what will reduce the token reward per inspector, what can discourage inspectors. When it happens, less active inspectors will participate so more inspections will be avaiable, what will decrease competition and increase token reward per inspector, what can bring back the interest of old or new inspectors, what can increase competition again and the cycle continues.

Even if it was a free or voluntary service, without the token earnings, there would probably have some people that would do it for free. They could do it by helping the system purpose/mission or to learn. Every test inspection I did over the last years testing the system I learned something new about agroforestry systems. I good path I see for the inspections is to attract people that want to learn about syntropic/agroforestry systems, people that want to become a farmer or students that can use this process to learn about ecology and reforestation. And earn some tokens in exchange for the service.

The proportionality was changed from 1:5 regenerator:inspector to 1:20. So now more inspectors vacancies are allowed.

To improve the pool balance, it was added 50 million tokens to the inspector pool. Also, the maximum amount of allowed inspections was reduced from 12 to 6. It is a trade between methodology precision to an easier operation. So the relation token/inspection will increase and inspectors will need to realize less of the work to get the same reward. PR #656

6.11 [Info] Inspectors are deactivated in their three give-ups

File(s): `InspectorRules.sol`

Description: The `isInspectorValid` function checks whether a given inspector address is still valid. It does this by comparing the number of `giveUps` associated with the inspector against the `MAX_GIVEUPS`. The function is stricter than intended, while the documentation states that an inspector can have three `giveUps`, the code allows only two before deactivation.

```
1  function isInspectorValid(address addr) public view returns (bool) {
2    return inspectors[addr].giveUps < MAX_GIVEUPS;
3  }
```

Recommendation(s): Consider allowing inspectors keep active with up to three giveUps.

Status: Fixed

Update from the client: The function isInspectorValid was updated to <= instead of only < to maintain the semantics of the text and allow 3 giveUps. Fixed at PR #655

```
1  function isInspectorValid(address addr) public view returns (bool) {
2    return inspectors[addr].giveUps <= MAX_GIVEUPS;
3  }
```

6.12 [Info] Lack of Incentives for Inspection Invalidation

File(s): InspectionRules.sol

Description: Inspectors submit their inspections by calling the `realizeInspection` function.

```
function realizeInspection(
  uint64 inspectionId,
  string memory proofPhotos,
  string memory justificationReport,
  uint32 treesResult,
  uint32 biodiversityResult
) external nonReentrant {
  // ...
  uint256 maxTreesForThisArea = uint256(regenerator.totalArea) * MAX_TREES_PER_SQM;
  require(treesResult <= maxTreesForThisArea, "Tree count exceeds density limit for this area");
  require(biodiversityResult <= MAX_BIODIVERSITY_RESULT, "Max result limit");

  // ...
  // @audit treesResult and biodiversityResult contributes to compute the ecological token value
  inspectionsTreesImpact += treesResult;
  inspectionsBiodiversityImpact += biodiversityResult;
  inspectorInspected[msg.sender][inspection.regenerator] = true;
  realizedInspectionsCount++;

  // ...
}
```

Users with voting rights can then call the function `addInspectionValidation` to invalidate a specific inspection. Once the required number of votes is reached, the inspection is invalidated. This mechanism is essential for preserving the integrity of the protocol, since inspection results directly impact the ecological value of the token.

```
function addInspectionValidation(uint64 id, string memory justification) external nonReentrant {
  require(!communityRules.isDenied(msg.sender), "User denied");
  require(bytes(justification).length <= MAX_TEXT_LENGTH, "Max characters reached");
  require(voteRules.canVote(msg.sender), "Not a voter");
  // ...
  inspection.validationsCount += 1;

  uint256 _votesToInvalidate = validationRules.votesToInvalidate();
  require(_votesToInvalidate >= 2, "Validation threshold cannot be less than 2");

  if (inspection.validationsCount >= _votesToInvalidate) {
    inspectionPenalized[id] = true;

    _invalidateInspection(inspection);

   //...
 // ...
}
```

However, as the community grows, both the number of votes required to invalidate an inspection and the overall number of inspections will increase. Currently, the protocol does not provide any incentives for users to participate in this invalidation process.

As a result, fraudulent inspections may remain as **valid** and continue to be used in the computation of the number of trees and biodiversity per token.

Recommendation(s): Introduce incentive mechanisms for users who vote to invalidate fraudulent inspections.

Status: Fixed

Update from the client: We solved the incentive problem at issue #1 with the creation of the ValidationPool. At this issue, we will focus on the solution for the problem described of the community unlimited growth and validation difficulty.

The system relies on the invalidation mechanism to clean wrong/fake inspections. The problem was that the validation mechanism, as pointed here, would keep increasing its difficulty as community grows with more resources/users and more votes, potentially leading to break the validation system. So we need to set a limit to solve it.

To mitigate the problem, we set a cap with a maximum number of active regenerators, to intentionally put a break on the system growth to solve this unlimited difficulty problem. So when the system reaches this maximum point, new regenerators registration will only be allowed when the old ones conclude the inspections lifecycle.

To make the calculations, a new mapping and variable was added. Now the system is registering how many regenerators have conclude the certification process.

```
  /// @notice The maximum number of active 'Regenerator' type users permitted in the system.
```

```
  uint256 public constant MAX_ACTIVE_REGENERATORS = 500000;

  /// @notice The total count of regenerators who have completed the certification process.
  uint256 public totalCertifiedRegenerators;

  /// @notice A mapping to track if a regenerator is a "certified regenerator", a user that has successfully
  /// completed the maximum inspections number, concluding system participation.
  mapping(address => bool) public certifiedRegenerators;
```

```
  function addRegenerator(
    uint32 totalArea,
    string memory name,
    string memory proofPhoto,
    string memory projectDescription,
    Coordinates[] calldata _coordinates
  ) external {
    // ..
    require(isRegistrationAllowed(), "Wait for vacancy: Max regenerators limit");
    // ..
  }
```

```
  /**
   * @notice Checks if new Regenerator registrations are allowed based on the dynamic count of active users.
   * @dev The number of active users is calculated as the total number of created Regenerators
   * minus the number of those who have completed their lifecycle.
   * @return bool True if registration is allowed, false otherwise.
   */
  function isRegistrationAllowed() public view returns (bool) {
    return communityRules.userTypesCount(USER_TYPE) - totalCertifiedRegenerators < MAX_ACTIVE_REGENERATORS;
  }
}
```

The certification variable is changed at the incrementInspection and decrementInspection functions. It is interesting to have this on the system, now we have a *bool* to show at the regenerators profile if they have completed or not.

```
  function decrementInspections(address addr) external mustBeAllowedCaller mustBeContractCall(inspectionRulesAddress) {
    // ..
    if (totalInspections == MAXIMUM_INSPECTIONS) {
      totalCertifiedRegenerators--;
      certifiedRegenerators[addr] = false;
    }

    regenerators[addr].totalInspections--;
  }
```

```
  function _incrementInspections(address addr) private returns (uint256) {
    // ..
    if (regenerator.totalInspections == MAXIMUM_INSPECTIONS) {
      certifiedRegenerators[addr] = true;
      totalCertifiedRegenerators++;

      emit RegeneratorCertified(addr);
    }

    return regenerator.totalInspections;
  }
```

A maximum number of required votes was also introduced to limit the percentage growth and create a mechanism that will drop the invalidation difficulty after reaching the point with 12k voters. Also, the timeBetweenVotes was reduced to approximately 2 hours. These changes were necessary to increase the number of users/resources the system will be able to process. Of course the mechanism will depend on the users participation on the governance mechanism to invalidate the inspections, at the end they need to call the functions. But now, with this cap mechanism and vote limit, it is mathematically possible for the validators to keep the integrity of the protocol.

```
  uint32 private constant VOTES_TO_INVALIDATE_LEVEL_3 = 360;


  /**
   * @notice Get how many validations is necessary to invalidate a user or resource.
   * @dev Calculates the required number of votes for invalidation based on the total number of registered voters in the
    ↪  system.
   * Calculation is based on the `votersCount` which includes researchers, developers, and contributors.
   * @return count Number of votes required for invalidation.
```

```
   */
  function votesToInvalidate() public view returns (uint256) {
    uint256 voters = communityRules.votersCount();
    // Threshold 1: Very early stage, requires a fixed number of 2 votes.
    if (voters < VOTERS_THRESHOLD_LEVEL_1) {
      return VOTES_TO_INVALIDATE_LEVEL_1;
    }
    // Threshold 2: Early stage, requires a fixed number of 5 votes.
    if (voters < VOTERS_THRESHOLD_LEVEL_2) {
      return VOTES_TO_INVALIDATE_LEVEL_2;
    }

    // Threshold 3: Mature stage, calculates votes based on a percentage of active voters.
    uint256 invalidationVotes = (voters * DYNAMIC_INVALIDATION_PERCENTAGE) / 100;
    uint256 requiredVotes = invalidationVotes + 1;

    // Threshold 4: Capped stage, returns a maximum votes level.
    if (requiredVotes > VOTES_TO_INVALIDATE_LEVEL_3) {
      return VOTES_TO_INVALIDATE_LEVEL_3;
    }

    return requiredVotes;
  }
```

Fixed at PR #661

6.13 [Info] Permanent Pool Misconfiguration Risk After Ownership Renouncement

File(s): RegenerationCredit.sol

Description: Each pool is expected to be initialized with a fixed amount defined in `TOTAL_POOL_TOKENS`. The `addContractPool` function currently allows the owner to deposit any arbitrary `_numTokens` value. The protocol is designed in the white paper that the owner will renounce ownership once it is deployed and initializations are applied. There is no way to correct the pool balances. If a pool is initialized with an incorrect deposit amount, it will not function properly, and the error becomes permanent.

```
function addContractPool(address _fundAddress, uint256 _numTokens) external onlyOwner returns (bool) {
    contractsPools[_fundAddress] = true;
    // @audit: Missing validation for _numTokens
    _transfer(msg.sender, _fundAddress, _numTokens);

    totalLocked_ += _numTokens;
    return true;
}
```

Recommendation: Ensure that `_numTokens` matches the required `TOTAL_POOL_TOKENS` for the specific pool when creating a new pool.

Status: Mitigated

Update from the client: This problem will be mitigated with a manual review of the contracts settings and balance after mainnet deploy. The same applies to other configurations such as allowedCallers and calls interfaces. For the review, it was missing to set the interfaces public so it can be easily checked. All interfaces were set to public to help.

```
  /// @notice The interface of the `CommunityRules` contract, used to interact with
  /// community-wide rules, user types, and invitation data.
  ICommunityRules public communityRules;

  /// @notice The interface of the `ContributorPool` contract, responsible for managing
  /// and distributing token rewards to contributors.
  IContributorPool public contributorPool;

  /// @notice The interface of the `ValidationRules` contract, which defines the rules
  /// and processes for validating or invalidating contributions.
  IValidationRules public validationRules;
```

PR #654

6.14 [Info] Unallocated tokens in `Eras` and `Epochs` will be completely locked in the contract

File(s): shared/Poolable.sol

Description: Tokens in a given pool for a given Epoch are calculated as :

```
function tokensPerEpoch(uint256 currentEpoch) public view returns (uint256) { //@audit there could be dust totalTokens
↪  remaining in the contract after certain epochs, rounding down.
    return totalTokens / (2 ** currentEpoch);
```

Subsequently, tokens per Era are calculated as :

```
function tokensPerEra(uint256 currentEpoch, uint256 halvingFactor) public view returns (uint256) {
    return tokensPerEpoch(currentEpoch) / halvingFactor; //@audit-note after every epoch per era tokens halves.
  }
```

Therefore, each epoch and era have a predetermined amount of tokens based on the total tokens allocated to a given pool. Per `era` and per `epoch` tokens are not carried forward. As a result, in the event there was no token allocation in a given `era` or `epoch` then those tokens will be stuck in the contract forever given there is no mechanism to withdraw them.

Recommendation(s): Consider implementing a logic to remove unallocated tokens from the pool contracts.

Status: Acknowledged.

Update from the client: This is the design choice and if tokens are unallocated they should remain locked in the contract.

6.15 [Info] Unclaimed Rewards Across Multiple Eras May Become Unrecoverable

File(s): `contracts/*Pool.sol`

Description: After each era ends, users can withdraw their rewards at any time. The system assumes that users will withdraw rewards from the previous era once a new era begins. However, the current design allows users to accumulate rewards across multiple eras and withdraw them later.

Missed Withdrawals Across Eras: A user may unintentionally skip withdrawals for some eras. If the user is later denied (e.g., an activist being penalized due to invitee misbehavior), they permanently lose access to all unclaimed rewards from previous eras.

On the other hand, if a user withdraws rewards after each era ends and is denied, they lose access only to the rewards from the current era.

Recommendation(s): Encourage users to withdraw rewards after each era ends to prevent future loss of their rewards.

Status: Acknowledged

Update from the client: Design choice. Users wil be encouraged to withdraw the rewards after each era. We could have developed a mechanism to set a limit amount of blocks to impose the need for withdraw in, or a way to send these tokens to the validationPool but for simplicity the design was maintained.

6.16 [Info] _canVoteRules function returns true when totalUsers is zero

File(s): VoteRules.sol

Description: The `_canVoteRules` function serves to check a user's eligibility to vote. However, in the specific case where `totalUsers==0`, it always returns `true`. As a result, the intended edge case that should prevent voting is never triggered.

```
  function _canVoteRules(uint256 totalTypeLevels, uint256 totalUsers, uint256 userLevels) private pure returns (bool) {

    // @audit When totalUsers == 0, the function returns true
    // Rule 1: Allow anyone to vote if the user type has few members.
    if (totalUsers <= INITIAL_USER_COUNT_THRESHOLD) return true;

    // @audit this edge case is never true
    // Edge case: If there are no users, no one can vote.
    if (totalUsers == 0) {
      return false;
    }

    // Rule 2: Check if the user's level is strictly greater than the average
    return userLevels * totalUsers > totalTypeLevels;
  }
```

However, `_canVoteRules` is only invoked by the canVote function. As shown in the code, it is impossible for `totalUsers == 0`, since verifying that `addr` is a voter and retrieving its `userType` ensures at least one user. The caller's own `addr` contributes to the count. Therefore, the logic mistake does not add risks for the current implementation.

```
function canVote(address addr) public view returns (bool) {
    require(communityRules.isVoter(addr), "Not a voter user");

    CommunityTypes.UserType userType = communityRules.getUser(addr);
    uint256 totalUsers = communityRules.userTypesTotalCount(userType);

    return _canVoteRules(_totalLevels(userType), totalUsers, _totalUserLevels(addr, userType));
}
```

Recommendation(s): For reuse of the private function in future improvements of the contract, consider only moving the edge case before the `Rule 1` condition.

Status: Fixed

Update from the client: The _canVoteRules function was reorganized to check the edge case before rule 1.

```
  function _canVoteRules(uint256 totalTypeLevels, uint256 totalUsers, uint256 userLevels) private pure returns (bool) {
    // Edge case: If there are no users, no one can vote.
    if (totalUsers == 0) {
      return false;
    }

    // Rule 1: Allow anyone to vote if the user type has few members.
    if (totalUsers <= INITIAL_USER_COUNT_THRESHOLD) return true;

    // Rule 2: Check if the user's level is strictly greater than the average
    return userLevels * totalUsers > totalTypeLevels;
  }
```

Fixed at PR #650

6.17 [Info] `canInvite` function returns true when `totalUsers` is zero

File(s): `Invitable.sol`

Description: The `canInvite` function serves to check a user's eligibility to invite others. However, in the specific case where `totalUsers == 0`, it always returns `true`. As a result, the intended edge case that should prevent invites is never triggered.

```
  function canInvite(uint256 totalLevels, uint256 totalUsers, uint256 userLevels) public pure returns (bool) {
    // Rule 1: Allow anyone to invite if the system has few users.
    if (totalUsers <= INITIAL_USER_COUNT_THRESHOLD) {
      return true;
    }

    @audit this condition will never be satisfied
    // If there are no users, no one can invite. This also prevents totalUsers from being zero in the calculation.
    if (totalUsers == 0) {
      return false;
    }

    // Rule 2: Check if the user's level is strictly greater than the average.
    return userLevels * totalUsers > totalLevels;// @audit review it again
  }
```

As shown in the code, it is not possible for `totalUsers == 0`, since verifying that `addr` is an inviter and retrieving its `userType` ensures at least one user. The caller's own `addr` contributes to the count. Therefore, the logic mistake does not add risks for the current implementation. The `canSendInvite` function is implemented in all roles contracts where users can send invites.

```
function canSendInvite(address addr) public view returns (bool) {
  Activist memory activist = activists[addr];

  // Return false if it is not an activist
  if (activist.id <= 0) return false;// @audit it is impossible negative

  // Calls the inherited `canInvite` function from `Invitable` to calculate eligibility.
  // This depends on total approved invites, total activist count, and the activist's pool level.
  return canInvite(approvedInvites, communityRules.userTypesTotalCount(USER_TYPE), activist.pool.level);
}
```

Recommendation(s): For reuse of the private function in future improvements of the contract, consider only moving the edge case before the `Rule 1` condition.

Recommendation(s):

Status: Fixed

Update from the client: The edge case was moved before the rule 1 at canInvite and canVote functions to check the zero value first.

```
  function canInvite(uint256 totalLevels, uint256 totalUsers, uint256 userLevels) public pure returns (bool) {
    // If there are no users, no one can invite. This also prevents totalUsers from being zero in the calculation.
    if (totalUsers == 0) {
      return false;
    }

    // Rule 1: Allow anyone to invite if the system has few users.
    if (totalUsers <= INITIAL_USER_COUNT_THRESHOLD) {
      return true;
    }

    // Rule 2: Check if the user's level is strictly greater than the average.
    return userLevels * totalUsers > totalLevels;
  }
```

Fixed at PR #651

6.18 [Best Practice] Event in `_denyUseris` never emitted

File(s): `ValidationRules.sol`

Description: The `_denyUser` function sets a user to `DENIED` in `CommunityRules` and removes their levels from the respective pools. The function below shows that when the `userType` matches with a type, it returns empty and the remaining code of the function is not executed.

```
function _denyUser(address userAddress) private {
  CommunityTypes.UserType userType = communityRules.getUser(userAddress);
  // ..
  communityRules.setToDenied(userAddress);
  // ..
  // Check for each user type and call their respective removePoolLevels function.
  if (userType == CommunityTypes.UserType.REGENERATOR) return regeneratorRules.removePoolLevels(userAddress);
  if (userType == CommunityTypes.UserType.INSPECTOR) return inspectorRules.removePoolLevels(userAddress, true);
  if (userType == CommunityTypes.UserType.DEVELOPER) return developerRules.removePoolLevels(userAddress);
  if (userType == CommunityTypes.UserType.RESEARCHER) return researcherRules.removePoolLevels(userAddress);
  if (userType == CommunityTypes.UserType.CONTRIBUTOR) return contributorRules.removePoolLevels(userAddress);
  if (userType == CommunityTypes.UserType.ACTIVIST) return activistRules.removePoolLevels(userAddress);

  // @audit this event is never emitted
  emit UserDenied(userAddress);
}
```

However, this event is duplicated because in `setToDenied` function, an event is emitted with the denied user address.

Recommendation(s): Remove this `event`, the `return`s and replace the `if` for `elseif`.

Status: Fixed

Update from the client: This event was removed since the communityRules.setToDenied() already emits a deny event. Fixed PR #652

7 Documentation Evaluation

Software documentation refers to the written or visual information that describes the functionality, architecture, design, and implementation of software. It provides a comprehensive overview of the software system and helps users, developers, and stakeholders understand how the software works, how to use it, and how to maintain it. Software documentation can take different forms, such as user manuals, system manuals, technical specifications, requirements documents, design documents, and code comments. Software documentation is critical in software development, enabling effective communication between developers, testers, users, and other stakeholders. It helps to ensure that everyone involved in the development process has a shared understanding of the software system and its functionality. Moreover, software documentation can improve software maintenance by providing a clear and complete understanding of the software system, making it easier for developers to maintain, modify, and update the software over time. Smart contracts can use various types of software documentation. Some of the most common types include:

- Technical whitepaper: A technical whitepaper is a comprehensive document describing the smart contract's design and technical details. It includes information about the purpose of the contract, its architecture, its components, and how they interact with each other;
- User manual: A user manual is a document that provides information about how to use the smart contract. It includes step-by-step instructions on how to perform various tasks and explains the different features and functionalities of the contract;
- Code documentation: Code documentation is a document that provides details about the code of the smart contract. It includes information about the functions, variables, and classes used in the code, as well as explanations of how they work;
- API documentation: API documentation is a document that provides information about the API (Application Programming Interface) of the smart contract. It includes details about the methods, parameters, and responses that can be used to interact with the contract;
- Testing documentation: Testing documentation is a document that provides information about how the smart contract was tested. It includes details about the test cases that were used, the results of the tests, and any issues that were identified during testing;
- Audit documentation: Audit documentation includes reports, notes, and other materials related to the security audit of the smart contract. This type of documentation is critical in ensuring that the smart contract is secure and free from vulnerabilities.

These types of documentation are essential for smart contract development and maintenance. They help ensure that the contract is properly designed, implemented, and tested, and they provide a reference for developers who need to modify or maintain the contract in the future.

Remarks about Sintrop documentation

Sintrop provided well-structured and comprehensive documentation on the functionality of its smart contracts. In addition, the team shared further clarifications through our communication channels and made themselves available for sync calls. This greatly facilitated the review process and offered valuable insights that enhanced our technical understanding of the project.

8 Test Suite Evaluation

8.1 Tests Output

```
> npx hardhat test

 ActivistPool
    after deploy
       must initial era equal one
    #getEra
      when access fields
         should have fields
    #nextEraIn
      when cant approve
         should return integer > zero
      when can approve
         should return integer < zero
    #addLevel
      with allowed caller
        when add level in era 1
          when activist have 0 levels in era 1
            when add level
               era 1 must have 2 level
               era 2 must have 0 level
               eraLevels must have 1 level to activist1
               eraLevels must have 1 level to activist2
          when activists have levels in era 1
            when add level
               era 1 must have 7 level
               era 2 must have 0 level
               eraLevels must have 3 level to activist1
               eraLevels must have 4 level to activist2
        when the same eventId is processed twice
           should revert the second transaction with the same eventId
           should have added the level only once
      without allowed caller
         should return error message
    #canWithdrawTimes
      when cant approve
         should return zero times
      when can approve 2 times
         should return two times
    #tokensPerEpoch
      when is epoch 1
         must return 200000000000000000000000000
      when is epoch 2
         must return 100000000000000000000000000
      when is epoch 3
         must return 50000000000000000000000000
      when is epoch 4
         must return 25000000000000000000000000
      when is epoch 5
         must return 12500000000000000000000000
      when is epoch 6
         must return 6250000000000000000000000
      when is epoch 7
         must return 3125000000000000000000000
      when is epoch 8
         must return 1562500000000000000000000
      when is epoch 9
         must return 781250000000000000000000
      when is epoch 10
         must return 390625000000000000000000
      when is epoch 15
         must return 12207031250000000000000
      when is epoch 20
         must return 381469726562500000000
    #tokensPerEra
      when is epoch 1
         must return 166666666666666666666666666
      when is epoch 2
         must return 833333333333333333333333
```

```
    when is epoch 3
       must return 41666666666666666666666
    when is epoch 4
       must return 208333333333333333333333
    when is epoch 5
       must return 104166666666666666666666
    when is epoch 6
       must return 52083333333333333333333
    when is epoch 7
       must return 26041666666666666666666
    when is epoch 8
       must return 13020833333333333333333
    when is epoch 9
       must return 651041666666666666666
    when is epoch 10
       must return 325520833333333333333
  #withdraw
    with allowed caller
      when can withdraw
        when is epoch 1
          when is era 1
            when total of levels in era is 6
              when activist1 have 3 levels in era 1
                 must withdraw 833333333333333333333333 tokens
              when activist1 have 6 levels in era 1
                 shoud withdraw 166666666666666666666666 tokens
                 shoud withdraw 0 tokens to activist2
              when activist2 have 3 levels in era 1
                 shoud withdraw 833333333333333333333333 tokens
          when is era 2
            when dont have withdraw from era 1
              when activist1 withdraw from era 1 and era 2
                 activist pool balance must be 366666666666666666666666668
                 activist1 balance must be 166666666666666666666666
                 activist1 balance in era 1 must be 833333333333333333333333
                 activist1 balance in era 2 must be 833333333333333333333333
                 must update eras
              when activist2 withdraw from era 1 and era 2
                 activist2 balance must be 166666666666666666666666
                 activist2 balance in era 1 must be 833333333333333333333333
                 activist2 balance in era 2 must be 833333333333333333333333
        when is epoch 2
          when is era 1
            when total of levels in era is 6
              when activist1 have 3 levels in era 1
                 must withdraw 833333333333333333333333 tokens
              when activist1 have 6 levels in era 1
                 shoud withdraw 166666666666666666666666 tokens
                 shoud withdraw 0 tokens to activist2
              when activist2 have 3 levels in era 1
                 shoud withdraw 833333333333333333333333 tokens
          when is era 2
            when dont have withdraw from era 1
              when activist1 withdraw from era 1 and era 2
                 activist pool balance must be 366666666666666666666666668
                 activist1 balance must be 166666666666666666666666
                 activist1 balance in era 1 must be 833333333333333333333333
                 activist1 balance in era 2 must be 833333333333333333333333
              when activist2 withdraw from era 1 and era 2
                 activist2 balance must be 166666666666666666666666
                 activist2 balance in era 1 must be 833333333333333333333333
                 activist2 balance in era 2 must be 833333333333333333333333
      when cant withdraw
         should return error message
    without allowed caller
       should return error message

ActivistRules
  #addActivist
    when is not an allowed user
       should return error message
    when is an allowed user
      when activist exists
         should return error
      when activist don\'t exist
```

```
        when max limit is not reached
           should create activist
           should increment activistCount
           should add created activist in userType contract as a ACTIVIST
        when max limit is reached
           should return error message
  #getActivist
    when activist is registered
       should return a activist
    when activist is registered
       should do not return a activist
  #addLevel
    with allowed caller
      when activist is registered
        when is not a valid user
           should return error when addRegeneratorLevel from other types
           should return error when addInspectorLevel from other types
        when current era of pool is 1
           add level to activist.pool.level
           add level to activisPool
           must increment approvedInvites
        when current era of pool is 2
           add level to activist.pool.level
           add level to era 2 activisPool
           must increment approvedInvites
      when activist is not registered
         do not add level to activist.pool.level
         do not add level to activisPool
    without allowed caller
       should return error message
  #withdraw
    when is a activist
      when is era 1
        when activist have levels
           should return error message
      when is era 2
        when activist have levels
          when have one activist
             activist to era 2
             activist balance must be
          when have two activist
             activist1 to era 2
             activist1 balance must be
             activist3 to era 2
             activist3 balance must be
        when activist do not have levels
          when have one activist
             activist to era 2
             activist balance must be
    when is not a activist
       should return error message
  #removePoolLevels
    when user is denied
       remove user levels from pool
       should keep user levels

Blockable
  #currentContractEra
    when don\'t have passed eras
       should return that be in era 1
    when have passed 1x the blocksPerEra
       should return that be in era 2
    when have passed 5x the blocksPerEra
       should return that be in era 6
  #currentEpoch
    when is era 1
       return currentEpoch equal 1
    when is era 6
       return currentEpoch equal 1
    when is era 12
       return currentEpoch equal 1
    when is era 13
       return currentEpoch equal 2
    when is era 24
       return currentEpoch equal 2
```

```
    when is era 25
       return currentEpoch equal 2
    when is era 36
       return currentEpoch equal 2
    when is era 37
       return currentEpoch equal 2
  #canWithdrawTimes
    when currentContractEra is equal currentUserEra
      with currentContractEra = 1 and currentUserEra = 1
         should can aprove zero times
      with currentContractEra = 5 and currentUserEra = 5
         should can aprove zero times
    when currentContractEra is bigger in one than currentUserEra
      with currentContractEra = 2 and currentUserEra = 1
         should can aprove one times
      with currentContractEra = 5 and currentUserEra = 4
         should can aprove one times
    when currentContractEra is bigger in two than currentUserEra
      with currentContractEra = 3 and currentUserEra = 1
         should can aprove 2 times
      with currentContractEra = 10 and currentUserEra = 8
         should can aprove 2 times
    when currentContractEra is bigger in five than currentUserEra
      with currentContractEra = 6 and currentUserEra = 1
         should can aprove 4 times
  #nextEraIn
    when user can approve
       should return negative blocks number
    when user can\'t approve
       should return positive blocks number
  #canWithdraw
    when currentUserEra is less than currentContractEra and currentUserEra don\'t have passed eraMax
       should return true
    when currentUserEra is equal currentContractEra
       should return false

Callable
   should return error when .newAllowedCaller and is not owner
   should add .newAllowedCaller when is owner
   should be able to add many callers .newAllowedCaller when is owner

CommunityRules
  #addUser
    with allowed caller
      when the user don\'t exist
         should add a user
         should increment usersCount
         should increment userTypesCount to regenerator
         should increment userTypesTotalCount to regenerator
         must emit UserRegistered
      when the user exists
         should return error message
      with UNDEFINED user type
         should return error message
      when enum correctly
        to Regenerator
           should add correct enum to regenerator
        to Inspector
           should add correct enum to inspector
        to researcher
           should add correct enum to researcher
        to developer
           should add correct enum to developer
        to contributor
           should add correct enum to contributor
        to activist
           should add correct enum to activist
        to supporter
           should add correct enum to supporter
      with proportionality invalid
        to inspector with proportionality 2
           should return error message
        to activist with proportionality 1
           should return error message
        to researcher with proportionality 1
```

```
          should return error message
      to developer with proportionality 1
          should return error message
      to contributor with proportionality 1
          should return error message
    when user was invited
      when try register as same user type of invitation
          should add a user
      when try register as another user type of invitation
          should return error message
    when user was not invited
        should return error message
  without allowed caller
     should return error message
#addInvitation
  with allowed caller
    when already invited
       should return error message
    when dont invited yet
       should invite
       must emit InvitationAdded
#getInvitation
   returns invitation
#usersCount
  without users
     should usersCount be zero
  with 1 user
     should usersCount be one
#newAllowedCaller
  with owner
     should add new allowed caller with success
  without owner
     should return error message
#addDelation
  when user1 and user2 is registed on system
    when user1 receive delation
       should add delation to user1
       should refer informer as user2
       should add have fields
       must emit DelationAdded
  when user1 (reported) is not registed on system
     should return error message
  when user2 (informer) is not registed on system
     should return error message
  when user2 (informer) is a supporter
     should return error message
  when a user tries to make multiple delations in a short period
    when a user tries to make multiple delations in a short period
       should revert if the cooldown period has not passed
       should succeed if the cooldown period has passed
#getUserDelations
  when user1 have 2 delations
     should return 2 delations
  when user1 have 0 delations
     should return 0 delations
#getUserTypeSettings
  when get to regenerator

  when get to contributor
     returns settings
  when get to inspector
     returns settings
  when get to activist
     returns settings
  when get to reseacher
     returns settings
  when get to developer
     returns settings
#votersCount
  when have 4 voters
    when all voters is not denied
       must returns 4 voters
    when some voters is  denied
       must return only valid voters
#isVoter
```

```
    when is regenerator
       must returns false
    when is activist
       must returns true
    when is developer
       must returns true
    when is contributor
       must returns true
    when is researcher
       must returns true
    when is denied
       must returns false
  #setToDenied
    with allowed user
       set denied to true
       user type must be the same
    with not allowed user
              should add contributor to era 2
             should withdraw all tokens from era
        when has two contrs in the era
          with same levels
            when Contributors is in era 1 and contract is in era 2
               should add contributor1 to era 2
               should add contributor2 to era 2
               contributor1 balance must be 8333333333333333333333
               contributor2 balance must be 8333333333333333333333
        when can withdraw only to one era and try withdraw again
           should return error message
        when can withdraw to two eras and try withdraw again
           should can withdraw in two eras
      when can\t withdraw tokens
         should return error message
    when is not contributor
       should return error message
  addContributionValidation
    when trying to vote on an already invalidated contribution
       should revert because the contribution is no longer valid
    with developer
      with valid contribution
        when contribution must be invalidated
           set valid field to false
           populate invalidatedAt field
           set maxPenalties to reseacher
           user type must be RESEARCHER yet
           must remove one pool level from current era
           must decrement contributionsCount in one
           should set contributionPenalized to true to prevent double penalties
        when contribution must not be invalidated
           valid field is true
           invalidatedAt is equal 0
           contributor totalPenalties is 0
           reseacher pool level is 1
           should keep contributionPenalized to false
      when contributor reach max maxPenalties
         user type must be DENIED
         should apply a penalty to the contributor\'s inviter
         should remove all pool levels for the denied contributor
         Remove invalidated contribution level from contributor
      with invalid contribution
        when current era is different from contribution created era
           should return error message
        when contribution is invalidated
           should return error message
        when contribution do not exists
           should return error message
    with researcher
      with valid contribution
        when contribution must be invalidated
           set valid field to false
           populate invalidatedAt field
           set maxPenalties to contributor
           user type must be CONTRIBUTOR yet
           must remove one pool level from current era
           do not must decrement contributionsTotalCount
           must decrement contributionsCount in one
```

```
        when contribution must not be invalidated
           valid field is true
           invalidatedAt is equal 0
           contributor totalPenalties is 0
           contributor pool level is 1
      when contributor reach max maxPenalties
         user type must be DENIED
      with invalid contribution
        when current era is different from contribution created era
           should return error message
        when contribution is invalidated
           should return error message
        when contribution do not exists
           should return error message
    with contributor
      with valid contribution
        when contribution must be invalidated
           set valid field to false
           populate invalidatedAt field
           set maxPenalties to contributor
           user type must be CONTRIBUTOR yet
           must remove one pool level from current era
           do not must decrement contributionsTotalCount
           must decrement contributionsCount in one
        when contribution must not be invalidated
           valid field is true
           invalidatedAt is equal 0
           contributor totalPenalties is 0
           contributor pool level is 1
      when contributor reach max maxPenalties
         user type must be DENIED
      with invalid contribution
        when current era is different from contribution created era
           should return error message
        when contribution is invalidated
           should return error message
        when contribution do not exists
           should return error message
    with activist
      with valid contribution
        when contribution must be invalidated
           set valid field to false
           populate invalidatedAt field
           set maxPenalties to reseacher
           user type must be CONTRIBUTOR yet
           must remove one pool level from current era
           must decrement contributionsCount in one
        when contribution must not be invalidated
           valid field is true
           invalidatedAt is equal 0
           contributioner totalPenalties is 0
           reseacher pool level is 1
      when contributioner reach max maxPenalties
         user type must be DENIED
      with invalid contribution
        when current era is different from contribution created era
           should return error message
        when contribution is invalidated
           should return error message
        when contribution do not exists
           should return error message
    without validator
       should return error message

DeveloperPool
  when deploy contract
     should initial be era equal one
  #getEra
    when access fields
       should have fields
  when check time to next approve
    when cant approve
       should return integer > zero
    when can approve
       should return integer < zero
```

```
  #addLevel
    with allowed caller
      when add level in era 1
        when developer have 0 levels in era 1
          when add level
             era 1 must have 2 level
             era 2 must have 0 level
             eraLevels must have 1 level to developer1
             eraLevels must have 1 level to developer2
        when developers have levels in era 1
          when add level
             era 1 must have 7 level
             era 2 must have 0 level
             eraLevels must have 3 level to developer1
             eraLevels must have 4 level to developer2
      when the same resource ID is processed twice
         should revert the second transaction with the same resource ID
         should have added the level only once
    without allowed caller
       should return error message
  when check how much times can approve
    when cant approve
       should return zero times
    when can approve 2 times
       should return two times
  #tokensPerEpoch
    when is epoch 1
       must return 200000000000000000000000000
    when is epoch 2
       must return 100000000000000000000000000
    when is epoch 3
       must return 50000000000000000000000000
    when is epoch 4
       must return 25000000000000000000000000
    when is epoch 5
       must return 12500000000000000000000000
    when is epoch 6
       must return 6250000000000000000000000
    when is epoch 7
       must return 3125000000000000000000000
    when is epoch 8
       must return 1562500000000000000000000
    when is epoch 9
       must return 781250000000000000000000
    when is epoch 10
       must return 390625000000000000000000
    when is epoch 15
       must return 122070312500000000000000
    when is epoch 20
       must return 38146972656250000000
  #tokensPerEra
    when is epoch 1
       must return 166666666666666666666666666
    when is epoch 2
       must return 833333333333333333333333
    when is epoch 3
       must return 416666666666666666666666666
    when is epoch 4
       must return 208333333333333333333333
    when is epoch 5
       must return 104166666666666666666666666
    when is epoch 6
       must return 52083333333333333333333
    when is epoch 7
       must return 260416666666666666666666
    when is epoch 8
       must return 130208333333333333333333
    when is epoch 9
       must return 651041666666666666666666
    when is epoch 10
       must return 325520833333333333333333
  #withdraw
    with allowed caller
      when can withdraw
        when is epoch 1
```

```
          when is era 1
            when total of levels in era is 6
              when dev1 have 3 levels in era 1
                 must withdraw 83333333333333333333333 tokens
                 must update get era
              when dev1 have 6 levels in era 1
                 shoud withdraw 166666666666666666666666 tokens
                 shoud withdraw 0 tokens to dev2
              when dev2 have 3 levels in era 1
                 shoud withdraw 83333333333333333333333 tokens
          when is era 2
            when dont have withdraw from era 1
              when dev1 withdraw from era 1 and era 2
                 dev pool balance must be 366666666666666666666666668
                 dev1 balance must be 166666666666666666666666
                 dev1 balance in era 1 must be 833333333333333333333333
                 dev1 balance in era 2 must be 833333333333333333333333
                 must update eras
              when dev2 withdraw from era 1 and era 2
                 dev2 balance must be 166666666666666666666666
                 dev2 balance in era 1 must be 833333333333333333333333
                 dev2 balance in era 2 must be 833333333333333333333333
        when is epoch 1
          when is era 1
            when total of levels in era is 6
              when dev1 have 3 levels in era 1
                 must withdraw 83333333333333333333333 tokens
              when dev1 have 6 levels in era 1
                 shoud withdraw 166666666666666666666666 tokens
                 shoud withdraw 0 tokens to dev2
              when dev2 have 3 levels in era 1
                 shoud withdraw 83333333333333333333333 tokens
          when is era 2
            when dont have withdraw from era 1
              when dev1 withdraw from era 1 and era 2
                 dev pool balance must be 366666666666666666666666668
                 dev1 balance must be 166666666666666666666666
                 dev1 balance in era 1 must be 833333333333333333333333
                 dev1 balance in era 2 must be 833333333333333333333333
              when dev2 withdraw from era 1 and era 2
                 dev2 balance must be 166666666666666666666666
                 dev2 balance in era 1 must be 833333333333333333333333
                 dev2 balance in era 2 must be 833333333333333333333333
      when cant withdraw
         should return error message
    without allowed caller
       should return error message

DeveloperRules
  .fields
     should have fields
  #addDeveloper
    when is not invited
       should return error message
    when is invited
      when developer exists
         should return error message
      when developer does not exist
        when max limit is not reached
           should add developer
           should increment developersCount after create developer
           should add created developer in userType contract as a DEVELOPER
           should add created developer with initial level equal 0
           should add created developer with initial currentEra equal currentContractEra
        when max limit is reached
           should return error message
  addReport
    with developer
      when have not waited time between works
         should return error message
      when have waited time between works
         should add report
      when don\'t have report
         add report
         increment reportsCount
```

```
        add level to developer
        add level to era
    when do not have security blocks to validator analysis
        should return error message
    when adding report to eras
        eras 1 must have 1 level
        eras 2 must have 1 level
  without developer
     should return error message
#getReport
   should have fields
#getReportsIds
   should have id associated
addReportValidation
  when trying to vote on an already invalidated report
     should revert because the report is no longer valid
  with developer
    with valid report
      when report must be invalidated
         set valid field to false
         populate invalidatedAt field
         set one penalty to developer
         should set reportPenalized to true to prevent double penalties
         user type must be DEVELOPER yet
         must remove one pool level from current era
         remove developer level report from developerRules
         must decrement reportsCount in one
      when report must not be invalidated
         valid field is true
         invalidatedAt is equal 0
         developer totalPenalties is 0
         developer pool level is 1
    when developer reach max maxPenalties
       user type must be DENIED
       should apply a penalty to the developer\'s inviter
       should remove all pool levels for the denied developer
       Remove invalidated report level from developer
    with invalid report
      when current era is different from report created era
         should return error message
      when report is invalidated
         should return error message
      when do not wait waitedTimeBetweenVotes
         should return error message
      when wait waitedTimeBetweenVotes
         should return error message
      when report do not exists
         should return error message
  with contributor
    with valid report
      when report must be invalidated
         set valid field to false
         populate invalidatedAt field
         set maxPenalties to contributor
         user type must be DEVELOPER yet
         user type must be DEVELOPER yet
         must remove one pool level from current era
         must decrement reportsCount in one
      when report must not be invalidated
         valid field is true
         invalidatedAt is equal 0
         contributor totalPenalties is 0
         contributor pool level is 1
    when developer reach maxPenalties
       user type must be DENIED
    with invalid report
      when current era is different from report created era
         should return error message
      when report is invalidated
         should return error message
      when report do not exists
         should return error message
  with researcher
    with valid report
      when report must be invalidated
```

```
            set valid field to false
            populate invalidatedAt field
            set maxPenalties to developer
            user type must be DEVELOPER yet
            must remove one pool level from current era
            do not must decrement reportsTotalCount
            must decrement reportsCount in one
        when report must not be invalidated
            valid field is true
            invalidatedAt is equal 0
            contributor totalPenalties is 0
            contributor pool level is 1
      when developer reach max maxPenalties
         user type must be DENIED
      with invalid report
        when current era is different from report created era
            should return error message
        when report is invalidated
            should return error message
        when report do not exists
            should return error message
    with activist
      with valid report
        when report must be invalidated
            set valid field to false
            populate invalidatedAt field
            set maxPenalties to developer
            user type must be DEVELOPER yet
            must remove one pool level from current era
            do not must decrement reportsTotalCount
            must decrement reportsCount in one
        when report must not be invalidated
            valid field is true
            invalidatedAt is equal 0
            contributor totalPenalties is 0
            contributor pool level is 1
      when contributor reach max maxPenalties
         user type must be DENIED
      with invalid report
        when current era is different from report created era
            should return error message
        when report is invalidated
            should return error message
        when report do not exists
            should return error message
    without validator
       should return error message
  #getDeveloper
     should return a developer
  #withdraw
    when is developer
      when can withdraw tokens
        when is unique developer in era with 1 level
          when Developer is in era 1 and contract is in era 2
             should add developer to era 2
             should withdraw all tokens from era
        when has two devs in the era
          with same levels
            when Developers is in era 1 and contract is in era 2
               should add developer1 to era 2
               should add developer2 to era 2
               developer1 balance must be 833333333333333333333333
               developer2 balance must be 833333333333333333333333
        when can withdraw only to one era and try withdraw again
            should return error message
        when can withdraw to two eras and try withdraw again
            should can withdraw in two eras
      when can\t withdraw tokens
         should return error message
    when is not developer
       should return error message
  #removePoolLevels
    when user is denied
       remove user levels from pool
       should not remove user levels from developer local level
```

```
InspectionRules
  #getInspection
    when inspection exists
       should return inspection
  #requestInspection
    with regenerator
      when have less than ALLOWED_INITIAL_REQUESTS
         should request inspection
      when have more than ALLOWED_INITIAL_REQUESTS
        when has request OPEN or ACCEPTED
           should return error message
        when don\'t has request OPEN or ACCEPTED
          when last request is recent
             should return error message
          when last request is not recent
             should request inspection
      when reached maximum inspections
         should return error
      #_afterRequestInspection
         initial status should be equal OPEN
         must set regenerator as regenerator address
         must set inspector as zero address
         initial regenerationScore should be equal zero
         should increment total of inspections
         should set to true regenerator pendingInspection
    with non regenerator
      when is not regenerator and try request inspection
         should return message error
  #acceptInspection
    with inspector
      when inspection exists
        when have not waited inspection delay time
           should return error message
        when do not have security blocks to validator analysis
          when nextEraIn is less than blocksToExpireAcceptedInspection
             should return error message
          when nextEraIn is bigger than blocksToExpireAcceptedInspection
             should return error message
        when have waited inspection delay time
           should accept with success
        when inspector has less than 3 giveups
           should accept with success
        when inspector has more than 3 giveups
           should return error message
        when never realized inspection from regenerator
          when inspection is OPEN
             accept inspection with success
             inspector must be inspectorAddress
             should increment inspector giveUps by 1
             Set last inspectionId to accepted inspection 1
          when inspection is not OPEN
             should return error message
          when have accepted other inspection
            when last inspection is not expired
               should return error message
            when last inspection is expired
               should accept inspection with success after blocksToExpireAcceptedInspection
            when have finished last inspection
               should return error when try to accept another inspection before wait blocks
               should accept inspection with success after finishing previous one
            when dont finished last inspection
               should return error message
        when already realized inspection from regenerator
           should return error message
      when inspection dont exists
         should return error message
      when regenerator is not valid
         should return error message
    with non inspector
       should return error message
  #realizeInspection
    with inspector
      when inspection exists
        when inspection is accepted
```

```
          when is accepted by inspector
            when inspection is expired
               should return error message
            when inspection is not expired
              when pass regenerationInspection equal 4 regenerationIndex size
                ._setActivistLevel
                  when regenerator do not win minimum inspection
                     Activist must do not win levels
                     Activist pool win 0 level to activist
                  when inspector do not win minimum inspection
                     Activist must do not win levels
                     Activist pool win 0 level to activist
                  when regenerator win minimum inspection
                     Activist must win 1 level
                     Activist pool win 1 level to activist
                  when inspector win minimum inspection
                     Activist must win 1 level
                     Activist pool win 1 level to activist
                  when regenerator and inspector win minimum inspection
                     Activist must win 1 level
                     Activist pool win 1 level to activist
                when check inspection
                   should change inspection status to INSPECTED
                   should set inspectionsTreesImpact
                   should set inspectionsBiodiversityImpact
                   populate inspection inspectedAt
                   should decrease inspector giveUps by 1
                   should add regenerationScore in regenerator
                   should set regenerator pendingInspection to false
                   should increment regenerator totalInspections
                   should increment inspector totalInspections
                   should increment realizedInspectionsCount
                when check inspection regenerationIndex
                  when tree result exceeds the dynamic density limit
                     should return error message
                  when biodiversity result exceeds the max limit
                     should return error message
                  when select REGENERATIVE_6
                     should add 64 regenerationScore to inspection
                  when select REGENERATIVE_5
                     should add 32 regenerationScore to inspection
                  when select REGENERATIVE_4
                     should add 16 regenerationScore to inspection
                  when select REGENERATIVE_3
                     should add 8 regenerationScore to inspection
                  when select REGENERATIVE_2
                     should add 4 regenerationScore to inspection
                  when select REGENERATIVE_1
                     should add 2 regenerationScore to inspection
                  when select NEUTRO
                     should add 0 regenerationScore to inspection
          when is accepted by other inspector
             should return error message
        when inspection is not accepted
           should return error message
      when inspection dont exists
         should return error message
    with non inspector
       should return error message
  #addInspectionValidation
    when trying to vote on an already invalidated inspection
       should revert because the inspection status is no longer INSPECTED
    with developer
      with valid inspection
        when receive 1 validation
           add validation
           should keep inspectionPenalized to false
        when have 2 validations (half of the validators)
          when inspection score is positive
             add validations
             decrement inspectionsTreesImpact
             decrement inspectionsTreesImpact
             inspection status INVALIDATED
             inspector receive 1 penalty
             should set inspectionPenalized to true to prevent double penalties
```

```
            remove regenerator regenerationScore
            decrement regenerator totalInspections
            decrement inspector totalInspections
            zero regeneratorPool era level score
            should decrement realizedInspectionsCount
      when inspector receive max penalties alloweds
         inspector type to DENIED
      when already voted in this inspection
         should return error message
    when inspection inspectedAtEra is passed
       should return error message
    when inspection is not inspected
       should return error message
  with contributor
    with valid inspection
      when receive 1 validation
         add validation
      when have 2 validations (half of the validators)
        when inspection score is positive
           add validations
           decrement inspectionsTreesImpact
           decrement inspectionsTreesImpact
           inspection status INVALIDATED
           inspector receive 1 penalty
           remove regenerator regenerationScore
           decrement regenerator totalInspections
           decrement inspector totalInspections
           zero regeneratorPool era level score
           should decrement realizedInspectionsCount
      when inspector receive max penalties alloweds
         inspector type to DENIED
         should apply a penalty to the inspector\'s inviter
         should remove all pool levels for the denied inspector
      when already voted in this inspection
         should return error message
    when inspection inspectedAtEra is passed
       should return error message
    when inspection is not inspected
       should return error message
  with activist
    with valid inspection
      when receive 1 validation
         add validation
      when have 2 validations (half of the validators)
        when inspection score is positive
           add validations
           decrement inspectionsTreesImpact
           decrement inspectionsTreesImpact
           inspection status INVALIDATED
           inspector receive 1 penalty
           remove regenerator regenerationScore
           decrement regenerator totalInspections
           decrement inspector totalInspections
           zero regeneratorPool era level score
           should decrement realizedInspectionsCount
      when inspector receive max penalties alloweds
         inspector type to DENIED
      when already voted in this inspection
         should return error message
    when inspection inspectedAtEra is passed
       should return error message
    when inspection is not inspected
       should return error message
  with researcher
    with valid inspection
      when receive 1 validation
         add validation
      when have 2 validations (votesToInvalidate)
        when inspection score is positive
           add validations
           decrement inspectionsTreesImpact
           decrement inspectionsTreesImpact
           inspection status INVALIDATED
           inspector receive 1 penalty
           remove regenerator regenerationScore
```

```
            decrement regenerator totalInspections
            decrement inspector totalInspections
            zero regeneratorPool era level score
            should decrement realizedInspectionsCount
        when inspector receive max penalties alloweds
           inspector type to DENIED
        when already voted in this inspection
           should return error message
      when inspection inspectedAtEra is passed
         should return error message
      when inspection is not inspected
         should return error message
    with non validator
       should return error message

InspectorPool
  after deploy
     must initial era equal one
  #getEra
    when access fields
       should have fields
  #nextEraIn
    when cant approve
       should return integer > zero
    when can approve
       should return integer < zero
  #addLevel
    with allowed caller
      when add level in era 1
        when inspector have 0 levels in era 1
          when add level
             era 1 must have 2 level
             era 2 must have 0 level
             eraLevels must have 1 level to inspector1
             eraLevels must have 1 level to inspector2
        when inspectors have levels in era 1
          when add level
             era 1 must have 7 level
             era 2 must have 0 level
             eraLevels must have 3 level to inspector1
             eraLevels must have 4 level to inspector2
      when the same resource ID is processed twice
         should revert the second transaction with the same resource ID
         should have added the level only once
    without allowed caller
       should return error message
  #canWithdrawTimes
    when cant approve
       should return zero times
    when can approve 2 times
       should return two times
  #tokensPerEpoch
    when is epoch 1
       must return 900000000000000000000000000
    when is epoch 2
       must return 450000000000000000000000000
    when is epoch 3
       must return 225000000000000000000000000
    when is epoch 4
       must return 112500000000000000000000000
    when is epoch 5
       must return 562500000000000000000000
    when is epoch 6
       must return 281250000000000000000000
    when is epoch 7
       must return 140625000000000000000000
    when is epoch 8
       must return 703125000000000000000
    when is epoch 9
       must return 351562500000000000000000
    when is epoch 10
       must return 175781250000000000000000
    when is epoch 15
       must return 5493164062500000000000
    when is epoch 20
```

```
      must return 171661376953125000000
  #tokensPerEra
    when is epoch 1
      must return 750000000000000000000000
    when is epoch 2
      must return 375000000000000000000000
    when is epoch 3
      must return 187500000000000000000000
    when is epoch 4
      must return 937500000000000000000000
    when is epoch 5
      must return 468750000000000000000000
    when is epoch 6
      must return 234375000000000000000000
    when is epoch 7
      must return 117187500000000000000000
    when is epoch 8
      must return 585937500000000000000000
    when is epoch 9
      must return 292968750000000000000000
    when is epoch 10
      must return 146484375000000000000000
  #withdraw
    with allowed caller
      when can withdraw
        when is epoch 1
          when is era 1
            when total of levels in era is 6
              when inspector1 have 3 levels in era 1
                 must withdraw 375000000000000000000000 tokens
              when inspector1 have 6 levels in era 1
                 shoud withdraw 120000000000000000000000 tokens
                 shoud withdraw 0 tokens to inspector2
              when inspector2 have 3 levels in era 1
                 shoud withdraw 375000000000000000000000 tokens
          when is era 2
            when dont have withdraw from era 1
              when inspector1 withdraw from era 1 and era 2
                 inspector pool balance must be 276000000000000000000000000
                 inspector1 balance must be 120000000000000000000000
                 inspector1 balance in era 1 must be 375000000000000000000000
                 inspector1 balance in era 2 must be 375000000000000000000000
                 must update eras
              when inspector2 withdraw from era 1 and era 2
                 inspector2 balance must be 120000000000000000000000
                 inspector2 balance in era 1 must be 375000000000000000000000
                 inspector2 balance in era 2 must be 375000000000000000000000
        when is epoch 2
          when is era 1
            when total of levels in era is 6
              when inspector1 have 3 levels in era 1
                 must withdraw 375000000000000000000000 tokens
              when inspector1 have 6 levels in era 1
                 shoud withdraw 120000000000000000000000 tokens
                 shoud withdraw 0 tokens to inspector2
              when inspector2 have 3 levels in era 1
                 shoud withdraw 375000000000000000000000 tokens
          when is era 2
            when dont have withdraw from era 1
              when inspector1 withdraw from era 1 and era 2
                 inspector pool balance must be 165000000000000000000000000
                 inspector1 balance must be 120000000000000000000000
                 inspector1 balance in era 1 must be 375000000000000000000000
                 inspector1 balance in era 2 must be 375000000000000000000000
              when inspector2 withdraw from era 1 and era 2
                 inspector2 balance must be 750000000000000000000000
                 inspector2 balance in era 1 must be 375000000000000000000000
                 inspector2 balance in era 2 must be 375000000000000000000000
      when cant withdraw
         should return error message
    without allowed caller
       should return error message

InspectorRules
  when access inspector fields
```

```
      should have fields
  when will create new inspector (.addInspector)
    when is allowed
      when inspector exists
         should return error
      when inspector don\'t exist
         should create inspector
         should be created with totalInspections equal zero
         should be created with giveUps equal zero
         should increment inspectorsCount after create inspector
         should add created inspector in userType contract as a INSPECTOR
  when will get inspector (.getInspector)
     should return a inspector
  #afterRealizeInspection
    with allowed caller
      .decreaseGiveUps
        must decreaseGiveUps
           should increment
      .incrementInspections
        when do not reached minimum inspections
           should increment
           should do not add level to pool
         when reached minimum inspections
            should add 1 level to pool
            should increment
    without allowed caller
       should return error when
  #afterAcceptInspection
    with allowed caller
      .incrementGiveUps
        must decreaseGiveUps
           should increment
      .markLastInspection
         must set lastAcceptedAt
         must set lastInspection
    without allowed caller
       should return error when
  #withdraw
    when is a inspector
      when have less then 3 inspections
         should return error
      when inspector is in era 1 and current era is 1
         should return error
      when inspector is in era 1 and current era is 2
        with one inspection
           withdraw 750000000000000000000000n tokens
        with two inspection
           withdraw 375000000000000000000000n tokens
    when is not a inspector
       should return error
  #addPenalty
    with allowed caller
       add penalty with success
    without allowed caller
       return erro message
  #totalPenalties
     return penalties

InvitationRules
  #invite
    when user already invited
       revert
    when user not registered and not invited
      when activist send invite
        when can invite
          when send to activist
            when have a previous invitation
              when is not recent
                 invite with success
              when is recent
                 revert
            when do not have a previous invitation
               invite with success
          when activist uses the generic invite function for restricted types
             should REVERT when trying to invite a Regenerator
```

```
            should REVERT when trying to invite an Inspector
            should PASS when trying to invite another Activist
        when send to inspector
          when have a previous invitation
            when is recent
               revert
          when do not have a previous invitation
             invite with success
        when send to regenerator
          when have a previous invitation
            when is recent
               revert
          when do not have a previous invitation
             invite with success
      cannot send invite
         returns message error
    when developer send invite
      when can invite
        when send to developer
          when have a previous invitation
            when is not recent
               invite with success
            when is recent
               revert
          when do not have a previous invitation
             invite with success
      can not send invite
         returns message error
    when contributor send invite
      when can invite
        when send to contributor
          when have a previous invitation
            when is not recent
               invite with success
            when is recent
               revert
          when do not have a previous invitation
             invite with success
      can not send invite
         returns message error
    when researcher send invite
      when can invite
        when send to researcher
          when have a previous invitation
            when is not recent
               invite with success
            when is recent
               revert
          when do not have a previous invitation
             invite with success
      can not send invite
         returns message error
    when supporter send invite
      when send to supporter
        when have a previous invitation
          when is not recent
             invite with success
          when is recent
             invite with success
        when do not have a previous invitation
           invite with success
  #inviteRegeneratorInspector
    when invite regenerator or inspector
      with activist
         should invite regenerator with success
         should invite inspector with success
      without activist
         revert
    when invite other types
       should revert
    when is recent
       should revert
  #onlyOwnerInvite
    when invite
      with owner
```

```
        invite with success
      with not owner
        revert
  InvitationRules #invite with penalty checks
    should revert if the inviter has reached the maximum penalty limit
    should succeed if the inviter has fewer than the maximum penalties
  InvitationRules #inviteRegeneratorInspector with penalty checks
    should revert if the inviter has reached the maximum penalty limit
    should succeed if the inviter has fewer than the maximum penalties

Poolable
  #tokensPerEra
    with valid epoch
      when current epoch is 1
        returns 166666666666666666666666 tokens
      when current epoch is 2
        returns 83333333333333333333333 tokens
      when current epoch is 3
        returns 41666666666666666666666 tokens
      when current epoch is 4
        returns 20833333333333333333333 tokens
      when current epoch is 5
        returns 10416666666666666666666 tokens
      when current epoch is 6
        returns 5208333333333333333333 tokens
      when current epoch is 7
        returns 2604166666666666666666 tokens
      when current epoch is 8
        returns 1302083333333333333333 tokens
      when current epoch is 9
        returns 651041666666666666666 tokens
      when current epoch is 10
        returns 325520833333333333333 tokens
  #tokensPerEpoch
    with valid epoch
      when current epoch is 1
        returns 200000000000000000000000000 tokens
      when current epoch is 2
        returns 100000000000000000000000000 tokens
      when current epoch is 3
        returns 50000000000000000000000000 tokens
      when current epoch is 4
        returns 25000000000000000000000000 tokens
      when current epoch is 5
        returns 12500000000000000000000000 tokens
      when current epoch is 6
        returns 6250000000000000000000000 tokens
      when current epoch is 7
        returns 3125000000000000000000000 tokens
      when current epoch is 8
        returns 1562500000000000000000000 tokens
      when current epoch is 9
        returns 781250000000000000000000 tokens
      when current epoch is 10
        returns 390625000000000000000000 tokens

RegenerationCredit
  .afterDeploy
    total suply should be equal to 150000000000000000000000000
    totalCertified should be equal zero
    totalLocked should be equal zero
    balance of contract owner should be equal to 150000000000000000000000000
  #addContractPool
    when totalTokens is bigger than tokens contract owner
      it should add totalLocked tokens
      it should add as a contract pool
      it should have balance
    when totalTokens is less than tokens contract owner
      must return erro message
  #transfer
    when user have tokens
      when a user transfer 1000000000000000000 sac token
        it should transfer when user has tokens
    when user doesn\'t have tokens
      must return erro message
```

```
    #approve
       it should transfer when user has tokens
    #transferFrom
      when user is approved to transfer from other address
        when tokenOwner is sufficient to transfer
           user2Address must receive 100000000000000000 tokens
           owner tokens must be decremented to 14999999990000000000000000000
        when tokenOwner is not sufficiente to transfer
           must return erro message
      when user is not approved to transfer from other address
         must return erro message
    #burnTokens
      when user have tokens
        when burn 1000000000000000000 tokens
           should burn when has tokens
           should add 1000000000000000000 to user certificate mapping
           should add 1000000000000000000 to totalCertified
        when burn another 1000000000000000000 tokens
           should burn when has tokens
           should increase 1000000000000000000 tokens to certificate mapping
           should increase totalCertified in 1000000000000000000
      when user does not have tokens
         must return error message
    #totalLocked
      when add contract pool
         it should set totalLocked to 750000000000000000000000000

  RegenerationCreditImpact
    totalTreesImpact
      when do not have inspections
         totalTreesImpact must be 0
      when have inspections
        when inspectionsTreesImpact is 0
           totalTreesImpact must be 0
        when inspectionsTreesImpact is not 0
          when have 1 inspection
            when inspection trees impact is 10
               must returnstotalTreesImpact equal 10
            when inspection trees impact is 25
               must returnstotalTreesImpact equal 32
          when have 5 inspections valid
             must returnstotalTreesImpact equal 174
          when have 3 inspections valids and two are of same regenerator
             must returnstotalTreesImpact equal 88
    totalCarbonImpact
      when do not have inspections
         totalCarbonImpact must be 0
      when have inspections
        when inspectionsTreesImpact is 0
           totalCarbonImpact must be 0
        when inspectionsTreesImpact is not 0
          when have 1 inspection
            when inspection trees impact is 10
               must returnstotalCarbonImpact equal 1000000
            when inspection trees impact is 25
               must returnstotalCarbonImpact equal 3200000
          when have 5 inspections valid
             must returnstotalCarbonImpact equal 17400000
          when have 3 inspections valids and two are of same regenerator
             must returnstotalCarbonImpact equal 8800000
    totalBiodiversityImpact
      when do not have inspections
         totalBiodiversityImpact must be 0
      when have inspections
        when inspectionsBiodiversityImpact is 0
           totalBiodiversityImpact must be 0
        when inspectionsBiodiversitympact is not 0
          when have 1 inspection
            when inspection biodiversity impact is 10
               must returns totalBiodiversityImpact equal 10
            when inspection biodiversity impact is 25
               must returns totalBiodiversityImpact equal 32
          when have 5 inspections valid
             must returns totalBiodiversityImpact equal 174
          when have 3 inspections valids and two are of same regenerator
```

```
          must returns totalBiodiversityImpact equal 88
  totalAreaImpact
    when have two regenerators
      when all regenerators are valids
         totalAreaImpact must be 2000
      when only one regenerator is valid
         totalAreaImpact must be 1000
  treesPerToken
    when do not have inspections
       treesPerToken must be 0
    when have inspections
      when inspectionsTreesImpact is 0
         treesPerToken must be 0
      when inspectionsTreesImpact is not 0
        when have 1 inspection
          when inspection trees impact is 10
            when do not have tokens totalLocked_
               must returns treesPerToken equal 6666666666
            when have token totalLocked_
               must returns treesPerToken equal 11111111111
          when inspection trees impact is 25
             must returnstotalTreesImpact equal 21333333333
        when have 5 inspections valid
           must returns treesPerToken equal 116000000000
        when have 3 inspections valids and two are of same regenerator
           must returns treesPerToken equal 58666666666
  carbonPerToken
    when do not have inspections
       carbonPerToken must be 0
    when have inspections
      when inspectionsTreesImpact is 0
         carbonPerToken must be 0
      when inspectionsTreesImpact is not 0
        when have 1 inspection
          when inspection trees impact is 10
            when do not have tokens totalLocked_
               must returns carbonPerToken equal 666666666666666
            when have token totalLocked_
               must returns carbonPerToken equal 1111111111111111
          when inspection trees impact is 25
             must returnstotalTreesImpact equal 2133333333333333
        when have 5 inspections valid
           must returns carbonPerToken equal 11600000000000000
        when have 3 inspections valids and two are of same regenerator
           must returns carbonPerToken equal 5866666666666666
  biodiversityPerToken
    when do not have inspections
       biodiversityPerToken must be 0
    when have inspections
      when inspectionsBiodiversityImpact is 0
         biodiversityPerToken must be 0
      when inspectionsBiodiversitympact is not 0
        when have 1 inspection
          when inspection biodiversity impact is 10
            when do not have tokens totalLocked_
               must returns biodiversityPerToken equal 6666666666
            when have tokens totalLocked_
               must returns biodiversityPerToken equal 11111111111
          when inspection biodiversity impact is 25
             must returns biodiversityPerToken equal 21333333333
        when have 5 inspections valid
           must returns biodiversityPerToken equal 116000000000
        when have 3 inspections valids and two are of same regenerator
           must returns biodiversityPerToken equal 58666666666
  areaPerToken
    when have two regenerators
      when all regenerators are valids
        when do not have tokens totalLocked_
           areaPerToken must be 1333333333333
        when have tokens totalLocked_
           areaPerToken must be 2222222222222
      when only one regenerator is valid
         areaPerToken must be 666666666666

RegenerationIndexRules
```

```
  #categories
     with category 1
     with category 2
  #getCategoryRegenerationIndexDescription
     with category 1
     with category 2
  #calculateScore
    when category and regeneration index exists
       calculate regenerationScore

RegeneratorPool
  #tokensPerEpoch
    when is epoch 1
       must return 375000000000000000000000000
    when is epoch 2
       must return 187500000000000000000000000
    when is epoch 3
       must return 93750000000000000000000000
    when is epoch 4
       must return 46875000000000000000000000
    when is epoch 5
       must return 23437500000000000000000000
    when is epoch 6
       must return 11718750000000000000000000
    when is epoch 7
       must return 5859375000000000000000000
    when is epoch 8
       must return 2929687500000000000000000
    when is epoch 9
       must return 1464843750000000000000000
    when is epoch 10
       must return 732421875000000000000000
  #tokensPerEra
    when is epoch 1
       must return 312500000000000000000000000
    when is epoch 2
       must return 156250000000000000000000000
    when is epoch 3
       must return 78125000000000000000000000
    when is epoch 4
       must return 39062500000000000000000000
    when is epoch 5
       must return 19531250000000000000000000
    when is epoch 6
       must return 9765625000000000000000000
    when is epoch 7
       must return 4882812500000000000000000
    when is epoch 8
       must return 2441406250000000000000000
    when is epoch 9
       must return 1220703125000000000000000
    when is epoch 10
       must return 610351562500000000000000
    when is epoch 15
       must return 22888183593750000000000
    when is epoch 20
       must return 715255737304687500000
  #withdraw
    with allowed caller
      when can approve
        when is epoch 1
          when totalScores is 0
             balanceOf must be 0
          when totalScores is 64
            when regenerator1 have 64 regenerationScore
               must withdraw 312500000000000000000000000 of tokens
               total locked must be 750000000000000000000000000 - 312500000000000000000000000 =
          when totalScores is 125
            when regenerator1 have 64 isaScore
               must withdraw 200000000000000000000000000 of tokens
            when regenerator2 have 45 isaScore
               must withdraw 112500000000000000000000000 tokens
        when is epoch 2
          when regenerator is in era 1 yet
            when totalScores is 0
```

```
              balanceOf must be 0
            when totalScores is 64
              when regenerator1 have 64 regenerationScore
                 must withdraw 312500000000000000000000000 of tokens
                 total locked must be 750000000000000000000000000 - 312500000000000000000000000
            when totalScores is 125
              when regenerator1 have 64 isaScore
                 must withdraw 200000000000000000000000000 of tokens
              when regenerator2 have 45 isaScore
                 must withdraw 112500000000000000000000000 tokens
      when can\t approve
         must return error message
    with not allowed caller
       must return error message
  #addLevel
    with allowed caller
      when add level in era 1
        when regenerator have 0 levels in era 1
          when add level
             era 1 must have 2 level
             era 2 must have 0 level
             eraLevels must have 1 level to regenerator1
             eraLevels must have 1 level to regenerator2
        when regenerators have levels in era 1
          when add level
             era 1 must have 7 level
             era 2 must have 0 level
             eraLevels must have 82 level to regenerator1
             eraLevels must have 4 level to regenerator2
      when add level in era 2
        when regenerators have levels in era 1
          when add level
             era 1 must have 80 level
             era 2 must have 40 level
             eraLevels must have 80 level to regenerator1
             eraLevels must have 4 level to regenerator2
      when the same inspection ID is processed twice
         should revert the second transaction with the same inspection ID
         should have added the levels only once
    without allowed caller
       should return error message

RegeneratorRules
  when access regenerator fields
     should have fields
  when will create a regenerator (.addRegenerator)
     should create regenerator
     should be created with totalRequest equal zero
     should be created with regenerationScore equal zero
     should be created with lastRequestAt equal zero
     should add coordinates to regenerator mapping
     should increment regeneratorsCount after create regenerator
     should add created regenerator in userType contract as a REGENERATOR
     should add totalArea to regenerationArea
  when coordinate points is invalid
    when coordinates is below 3
       should return error when try create regenerator without valid coordinates
    when coordinates is bigger than 10
       should return error when try create regenerator without valid coordinates
  when coordinate values are invalid
     should revert if coordinates contain duplicates
     should revert if latitude is out of range (> 90)
     should revert if longitude is out of range (> 180)
     should revert if a coordinate string contains invalid characters
     should revert if a coordinate string contains multiple dots
  when totalArea is below 500
     should return error
  when totalArea is over 500.000
     should return error
  when regenerator alredy exists
     should return error when try create same regenerator
  #afterRequestInspection
    with allowed caller
       set pendingInspection to true
       set lastRequestAt
```

```
    with not allowed caller
       return message error
  #afterRealizeInspection
    with allowed user
      .setRegenerationScore
       when dont have regenerators sustainable
         when have 1 regenerator
           when new score + regenerator score is smaller than limit score
              regenerator regeneration score must be 96
         when have more than one regenerator
           when new score + regenerator A score is smaller than limit score
              regenerator regeneration score must be 67
       when regenerator have reached minimum inspections
         when is era 1
           when already have 50 levels in regenerator contract
             when receives more 25 levels
               when is not in the pool yet
                  set 75 levels to era 1 pool
                  regenerator regenerationScore must be 75
               when already in the pool
                  set 100 levels to era 1 pool
                  regenerator regenerationScore must be 100
         when is era 2
           when already have 50 levels in regenerator contract
             when receives more 50 levels
                set 50 levels to era 2 pool
                regenerator regenerationScore must be 100
      .incrementInspections
        incrementInspections
    with not allowed user
       should return error message
  #withdraw
    with regenerator
      when can approve #blockable
        when regenerator have minimum inspections
          when levels in era is 100
            when regenerator have regenerationScore 50
               regenerator A must withdraw 37500000000000000000000000n tokens
               regenerator B must withdraw 37500000000000000000000000n tokens
               regenerator A current era must be incremented
               regenerator B current era must be incremented
            when regenerator have regenerationScore 50
               must withdraw 75000000000000000000000000n tokens
               regenerator current era must be increment
        when regenerator dont have minimum inspections
           should return error message
      when cant approve #blockable
         should return error message
    with not regenerator
       should return error message
  #regeneratorPoolEra
    when pool is in era 1
       return era equal 1
    when pool is in era 2
       return era equal 1
  #updateAreaPhoto
    without regenerator
       should return error
    with regenerator
       should update photo
  #getCoordinates
    without regenerator
       should return coordinates arrayList

ResearcherPool
  after deploy
     must initial era equal one
  #getEra
    when access fields
       should have fields
  #nextEraIn
    when cant approve
       should return integer > zero
    when can approve
       should return integer < zero
```

```
#addLevel
  with allowed caller
    when add level in era 1
      when researcher have 0 levels in era 1
        when add level
           era 1 must have 2 level
           era 2 must have 0 level
           eraLevels must have 1 level to researcher1
           eraLevels must have 1 level to researcher2
      when researchers have levels in era 1
        when add level
           era 1 must have 7 level
           era 2 must have 0 level
           eraLevels must have 3 level to researcher1
           eraLevels must have 4 level to researcher2
    when the same resource ID is processed twice
       should revert the second transaction with the same resource ID
       should have added the level only once
  without allowed caller
     should return error message
#canWithdrawTimes
  when cant approve
     should return zero times
  when can approve 2 times
     should return two times
#tokensPerEpoch
  when is epoch 1
     must return 200000000000000000000000000
  when is epoch 2
     must return 100000000000000000000000000
  when is epoch 3
     must return 50000000000000000000000000
  when is epoch 4
     must return 25000000000000000000000000
  when is epoch 5
     must return 12500000000000000000000000
  when is epoch 6
     must return 6250000000000000000000000
  when is epoch 7
     must return 3125000000000000000000000
  when is epoch 8
     must return 1562500000000000000000000
  when is epoch 9
     must return 781250000000000000000000
  when is epoch 10
     must return 390625000000000000000000
  when is epoch 15
     must return 122070312500000000000000
  when is epoch 20
     must return 38146972656250000000
#tokensPerEra
  when is epoch 1
     must return 166666666666666666666666666
  when is epoch 2
     must return 833333333333333333333333
  when is epoch 3
     must return 416666666666666666666666666
  when is epoch 4
     must return 208333333333333333333333
  when is epoch 5
     must return 104166666666666666666666666
  when is epoch 6
     must return 52083333333333333333333
  when is epoch 7
     must return 260416666666666666666666
  when is epoch 8
     must return 130208333333333333333333
  when is epoch 9
     must return 651041666666666666666666
  when is epoch 10
     must return 325520833333333333333333
#withdraw
  with allowed caller
    when can withdraw
      when is epoch 1
```

```
          when is era 1
            when total of levels in era is 6
              when researcher1 have 3 levels in era 1
                 must withdraw 8333333333333333333333 tokens
              when researcher1 have 6 levels in era 1
                 shoud withdraw 166666666666666666666666 tokens
                 shoud withdraw 0 tokens to researcher2
              when researcher2 have 3 levels in era 1
                 shoud withdraw 8333333333333333333333 tokens
          when is era 2
            when dont have withdraw from era 1
              when researcher1 withdraw from era 1 and era 2
                 researcher pool balance must be 366666666666666666666666668
                 researcher1 balance must be 166666666666666666666666
                 researcher1 balance in era 1 must be 833333333333333333333333
                 researcher1 balance in era 2 must be 833333333333333333333333
                 must update eras
              when researcher2 withdraw from era 1 and era 2
                 researcher2 balance must be 166666666666666666666666
                 researcher2 balance in era 1 must be 833333333333333333333333
                 researcher2 balance in era 2 must be 833333333333333333333333
        when is epoch 2
          when is era 1
            when total of levels in era is 6
              when researcher1 have 3 levels in era 1
                 must withdraw 8333333333333333333333 tokens
              when researcher1 have 6 levels in era 1
                 shoud withdraw 166666666666666666666666 tokens
                 shoud withdraw 0 tokens to researcher2
              when researcher2 have 3 levels in era 1
                 shoud withdraw 833333333333333333333333 tokens
          when is era 2
            when dont have withdraw from era 1
              when researcher1 withdraw from era 1 and era 2
                 researcher pool balance must be 366666666666666666666666668
                 researcher1 balance must be 166666666666666666666666
                 researcher1 balance in era 1 must be 833333333333333333333333
                 researcher1 balance in era 2 must be 833333333333333333333333
              when researcher2 withdraw from era 1 and era 2
                 researcher2 balance must be 166666666666666666666666
                 researcher2 balance in era 1 must be 833333333333333333333333
                 researcher2 balance in era 2 must be 833333333333333333333333
      when cant withdraw
         should return error message
    without allowed caller
       should return error message

ResearcherRules
  #addResearcher
    when is not an allowed user
       should return error message
    when is an allowed user
      when researcher already exists
         should return error
      when researcher don\'t exist
        when max limit is not reached
           create researcher
           increment researcherCount after create researcher
           add created researcher in userType contract as a RESEARCHER
           add created researcher with 0 published researches
           add created researcher with 0 published items
           add created researcher with publishedMethod = true
        when max limit is reached
           should return error message
  #getResearcher
     return a researcher
  #getResearcher
    when researcher exists
       return true
    when researcher don\'t exist
       return false
  #withdraw
    when is a researcher
      when researcher is in era 1 and current era is 1
         should return error
```

```
      when researcher is in era 1 and current era is 2
        with one researches
           withdraw 16666666666666666666666666 tokens
        with one researches
           withdraw 833333333333333333333333 tokens
    when is not a researcher
       should return error
  #addResearch
    when is not a researcher
       should return error
    when is a researcher
      when have time to validator analysis
        when have waited time between researches
           add a research
           add 1 to researcher publishedResearches
           add 1 to researcher pool level
           add 1 to researcher pool eraLeves
           dont add to researcher pool eraLeves of other era
          when is next era
             add +1 to researcher pool eraLeves of era 2
             add +1 to researcher pool level
             dont add to researcher pool eraLeves of other era
        when have not waited time between works
           should return error message
      when do not have time to validator analysis
         should return error message
  #getResearch
     should have fields
  addResearchValidation
    when trying to vote on an already invalidated research
       should revert because the research is no longer valid
    with researcher
      with valid contribution
        when research must be invalidated
           set valid field to false
           populate invalidatedAt field
           set maxPenalties to reseacher
           user type must be RESEARCHER yet
           must remove one pool level from current era
           must decrement researchesCount in one
        when research must not be invalidated
           valid field is true
           invalidatedAt is equal 0
           researcher totalPenalties is 0
           reseacher pool level is 1
      when researcher reach max maxPenalties
         user type must be DENIED
         Remove invalidated research level from researcher
      with invalid research
        when current era is different from contribution created era
           should return error message
        when contribution is invalidated
           should return error message
        wwhen do not wait waitedTimeBetweenVotes
           should return error message
        wwhen wait waitedTimeBetweenVotes
           should return error message
        when contribution do not exists
           should return error message
    with developer
      with valid contribution
        when research must be invalidated
           set valid field to false
           populate invalidatedAt field
           set maxPenalties to reseacher
           user type must be RESEARCHER yet
           must remove one pool level from current era
           must decrement researchesCount in one
           should set researchPenalized to true to prevent double penalties
        when research must not be invalidated
           valid field is true
           invalidatedAt is equal 0
           researcher totalPenalties is 0
           reseacher pool level is 1
           should keep researchPenalized to false
```

```
      when researcher reach max maxPenalties
         user type must be DENIED
         should apply a penalty to the researcher\'s inviter
         should remove all pool levels for the denied researcher
         Remove invalidated research level from researcher
      with invalid research
        when current era is different from research created era
           should return error message
        when research is invalidated
           should return error message
        when research do not exists
           should return error message
    with contributor
      with valid contribution
        when research must be invalidated
           set valid field to false
           populate invalidatedAt field
           set maxPenalties to reseacher
           user type must be RESEARCHER yet
           must remove one pool level from current era
           must decrement researchesCount in one
        when research must not be invalidated
           valid field is true
           invalidatedAt is equal 0
           researcher totalPenalties is 0
           reseacher pool level is 1
      when researcher reach maxPenalties
         user type must be DENIED
         Remove invalidated research level from researcher
      with invalid research
        when current era is different from contribution created era
           should return error message
        when contribution is invalidated
           should return error message
        when contribution do not exists
           should return error message
    with activist
      with valid contribution
        when research must be invalidated
           set valid field to false
           populate invalidatedAt field
           set maxPenalties to reseacher
           user type must be RESEARCHER yet
           must remove one pool level from current era
           must decrement researchesCount in one
        when research must not be invalidated
           valid field is true
           invalidatedAt is equal 0
           researcher totalPenalties is 0
           reseacher pool level is 1
      when researcher reach max maxPenalties
         user type must be DENIED
         Remove invalidated research level from researcher
      with invalid research
        when current era is different from contribution created era
           should return error message
        when contribution is invalidated
           should return error message
        when contribution do not exists
           should return error message
    without validator
       should return error message
  #removePoolLevels
    when user is to denied
       remove user levels from pool
       remove user levels from researcher
  #addCalculatorItem
    when is not a researcher
       should return error
    when is a researcher
      when have waited time between calculatorItems
         add an calculatorItem
         should add researcher publishedItems
      when have not waited time between calculatorItems
         should return error message
```

```
      when carbon is out of range
         should return error message when set to zero
         should return error message when set over 1 ton
  #addEvaluationMethod
    when is not a researcher
       should return error
    when is a researcher
      when did not publish a method before
         add an evaluationMethod
         must set researcher bool to false
      when publish second method
         should return error message

SupporterRules
  #addSupporter
    when supporter exists
       should return error
    when supporter don\'t exist
       create supporter
       increment supporterCount
       add created supporter in userType contract as a SUPPORTER
  #updateProfilePhoto
    without supporter
       should return error
    with supporter
       should update photo
  #getSupporter
     return a supporter
  #offset
    when msg.sender is SUPPORTER
      when amount is valid
        when calculatorItemId exists
          when SUPPORTER was invited
            when burn 100000000000000000 tokens
               calculatorItemCertificates to item 1 must be 95000000000000000
               must add offset amount
               must add offset total count
               must add offset supporter count
               must pay comission
          when SUPPORTER wasn't invited
            when burn 100000000000000000 tokens
               calculatorItemCertificates to item 1 must be 1000000000000000000n
            when burn 500000000000000000 tokens
               calculatorItemCertificates to item 1 must be 5000000000000000000n
            when burn multiple times
               calculatorItemCertificates must sum all offsets
        when calculatorItemId does not exists
          when burn 100000000000000000 tokens
            calculatorItemCertificates to item 10 must be 0
      when amount is invalid
        without allowance
           should return error
        without minimum amount
           should return error
  #declareReductionCommitment
    when is supporter
      when calculatorItem exists
         return a supporter
         must add reduction item supporter count
      when try to declared the same item twice
         should return error
      when calculatorItem does not exists
         should return error
    when is not a supporter
       return a supporter

ValidationRules
  #addUserValidation
    when caller is validator
      when user already is denied
         should return error
      when user is not denied
        when user validations count is less than votesToInvalidate
          with developer
            when total users is less than 5
```

```
            should add validation
            user type must be the same
            inviter must not get penalty
          when total users is bigger than 5
            when user levels is less than total users levels avg
              should return error
            when user levels is bigger than total users levels avg
              should add validation
              user type must be the same
        with contributor
          when total users is less than 5
            should add validation
            user type must be the same
            inviter must not get penalty
          when total users is bigger than 5
            when user levels is less than total users levels avg
              should return error
            when user levels is bigger than total users levels avg
              should add validation
              user type must be the same
        with researcher
          when total users is less than 5
            should add validation
            user type must be the same
            inviter must not get penalty
          when total users is bigger than 5
            when user levels is less than total users levels avg
              should return error
            when user levels is bigger than total users levels avg
              should add validation
              user type must be the same
        with activist
          when total users is less than 5
            should add validation
            user type must be the same
            inviter must not get penalty
          when total users is bigger than 5
            when user levels is less than total users levels avg
              should return error
            when user levels is bigger than total users levels avg
              should add validation
              user type must be the same
      when user validations count is equal or bigger than votesToInvalidate
        when current era is 1
          with regenerator
            should add validation
            user type must be denied
            inviter must get 1 penalty
            remove user levels from pool
            remove user regenerationScore from regenerator
            userTypesCount must be decremented
            regenerationArea should be decremented
            must emit DeniedUserEvent
          with inspector
            should add validation
            user must be denied
            remove user levels from pool
            do not remove user levels from inspector
            userTypesCount must be decremented
          with contributor
            should add validation
            user must be denied
            remove user levels from pool
            do not remove user levels from contributor
            userTypesCount must be decremented
          with developer
            should add validation
            user must be denied
            remove user levels from pool
            do not remove user levels from developer
            userTypesCount must be decremented
          with researcher
            should add validation
            user must be denied
            remove user levels from pool
```

```
                do not remove user levels from researcher
                userTypesCount must be decremented
             with activist
                should add validation
                user must be denied
                remove user levels from pool
                do not remove user levels from activist
                userTypesCount must be decremented
          when current era is 2
            when add validation
                should add validation
                user type must be denied
    when caller is not validator
       should return error
    when validator already voted to user
       should return error
    when do not wait waitedTimeBetweenVotes
       should return error
    when do not wait waitedTimeBetweenVotes
       should return error
    when is not a registered user
       should return error
    when regenerator _canBeValidated
      when reached REGENERATOR_VALIDATION_IMMUNITY_THRESHOLD
         must returns error message
      when do not reached REGENERATOR_VALIDATION_IMMUNITY_THRESHOLD
         should add validation
         user type must be the same
  #votesToInvalidate
    when voters is less then 10
       returns 2
    when voters is 20
       returns 2
    when votersCount is less than 166
       returns 5
    when votersCount is 168
       returns 6
    when votersCount is 249
       returns 8
    when votersCount is 255
       returns 8
    when votersCount is 600
       returns 19
    when votersCount is 1200
       returns 37
    when votersCount is 2500
       returns 76
    when votersCount is 5000
       returns 151
    when votersCount is 10000
       returns 301
    when votersCount is 20000
       returns 601
    wwhen votersCount is 35000
       returns 1051
    when votersCounts is 64000
       returns 1921

VoteRules
  #canVote
    when is a voter
      with activist
        when totalLevels is 4, totalUsers is 4 and userLevels is 1
           returns true
        when totalLevels is 10, totalUsers is 10 and userLevels is 1
           returns false
        when totalLevels is 60, totalUsers is 30 and userLevels is 1
           returns false
        when totalLevels is 60, totalUsers is 30 and userLevels is 3
           returns true
        when totalLevels is 6000, totalUsers is 600 and userLevels is 50
           returns true
        when totalLevels is 0, totalUsers is 10 and userLevels is 0
           returns false
        when totalLevels is 50000, totalUsers is 5000 and userLevels is 10
```

```
            returns false
          when totalLevels is 50000, totalUsers is 5000 and userLevels is 20
            returns true
        with contributor
          when totalLevels is 4, totalUsers is 4 and userLevels is 1
            returns true
          when totalLevels is 10, totalUsers is 10 and userLevels is 1
            returns false
          when totalLevels is 60, totalUsers is 30 and userLevels is 1
            returns false
          when totalLevels is 60, totalUsers is 30 and userLevels is 3
            returns true
          when totalLevels is 6000, totalUsers is 600 and userLevels is 50
            returns true
          when totalLevels is 0, totalUsers is 10 and userLevels is 0
            returns false
          when totalLevels is 50000, totalUsers is 5000 and userLevels is 10
            returns false
          when totalLevels is 50000, totalUsers is 5000 and userLevels is 20
            returns true
        with developer
          when totalLevels is 4, totalUsers is 4 and userLevels is 1
            returns true
          when totalLevels is 10, totalUsers is 10 and userLevels is 1
            returns false
          when totalLevels is 60, totalUsers is 30 and userLevels is 1
            returns false
          when totalLevels is 60, totalUsers is 30 and userLevels is 3
            returns true
          when totalLevels is 6000, totalUsers is 600 and userLevels is 50
            returns true
          when totalLevels is 0, totalUsers is 10 and userLevels is 0
            returns false
          when totalLevels is 50000, totalUsers is 5000 and userLevels is 10
            returns false
          when totalLevels is 50000, totalUsers is 5000 and userLevels is 20
            returns true
        with researcher
          when totalLevels is 4, totalUsers is 4 and userLevels is 1
            returns true
          when totalLevels is 10, totalUsers is 10 and userLevels is 1
            returns false
          when totalLevels is 60, totalUsers is 30 and userLevels is 1
            returns false
          when totalLevels is 60, totalUsers is 30 and userLevels is 3
            returns true
          when totalLevels is 6000, totalUsers is 600 and userLevels is 50
            returns true
          when totalLevels is 0, totalUsers is 10 and userLevels is 0
            returns false
          when totalLevels is 50000, totalUsers is 5000 and userLevels is 10
            returns false
          when totalLevels is 50000, totalUsers is 5000 and userLevels is 20
            returns true
      when is not a voter
        should return error message


  1268 passing (1m)
```

9 About Nethermind

Nethermind is a Blockchain Research and Software Engineering company. Our work touches every part of the web3 ecosystem - from layer 1 and layer 2 engineering, cryptography research, and security to application-layer protocol development. We offer strategic support to our institutional and enterprise partners across the blockchain, digital assets, and DeFi sectors, guiding them through all stages of the research and development process, from initial concepts to successful implementation.

We offer security audits of projects built on EVM-compatible chains and Starknet. We are active builders of the Starknet ecosystem, delivering a node implementation, a block explorer, a Solidity-to-Cairo transpiler, and formal verification tooling. Nethermind also provides strategic support to our institutional and enterprise partners in blockchain, digital assets, and decentralized finance (DeFi). In the next paragraphs, we introduce the company in more detail.

Blockchain Security: At Nethermind, we believe security is vital to the health and longevity of the entire Web3 ecosystem. We provide security services related to Smart Contract Audits, Formal Verification, and Real-Time Monitoring. Our Security Team comprises blockchain security experts in each field, often collaborating to produce comprehensive and robust security solutions. The team has a strong academic background, can apply state-of-the-art techniques, and is experienced in analyzing cutting-edge Solidity and Cairo smart contracts, such as ArgentX and StarkGate (the bridge connecting Ethereum and StarkNet). Most team members hold a Ph.D. degree and actively participate in the research community, accounting for 240+ articles published and 1,450+ citations in Google Scholar. The security team adopts customer-oriented and interactive processes where clients are involved in all stages of the work.

Blockchain Core Development: Our core engineering team, consisting of over 20 developers, maintains, improves, and upgrades our flagship product - the Nethermind Ethereum Execution Client. The client has been successfully operating for several years, supporting both the Ethereum Mainnet and its testnets, and now accounts for nearly a quarter of all synced Mainnet nodes. Our unwavering commitment to Ethereum's growth and stability extends to sidechains and layer 2 solutions. Notably, we were the sole execution layer client to facilitate Gnosis Chain's Merge, transitioning from Aura to Proof of Stake (PoS), and we are actively developing a full-node client to bolster Starknet's decentralization efforts. Our core team equips partners with tools for seamless node set-up, using generated docker-compose scripts tailored to their chosen execution client and preferred configurations for various network types.

DevOps and Infrastructure Management: Our infrastructure team ensures our partners' systems operate securely, reliably, and efficiently. We provide infrastructure design, deployment, monitoring, maintenance, and troubleshooting support, allowing you to focus on your core business operations. Boasting extensive expertise in Blockchain as a Service, private blockchain implementations, and node management, our infrastructure and DevOps engineers are proficient with major cloud solution providers and can host applications in-house or on clients' premises. Our global in-house SRE teams offer 24/7 monitoring and alerts for both infrastructure and application levels. We manage over 5,000 public and private validators and maintain nodes on major public blockchains such as Polygon, Gnosis, Solana, Cosmos, Near, Avalanche, Polkadot, Aptos, and StarkWare L2. Sedge is an open-source tool developed by our infrastructure experts, designed to simplify the complex process of setting up a proof-of-stake (PoS) network or chain validator. Sedge generates docker-compose scripts for the entire validator set-up based on the chosen client, making the process easier and quicker while following best practices to avoid downtime and being slashed.

Cryptography Research: At Nethermind, our cryptography Research team conducts cutting-edge internal research and collaborates closely with external partners on cryptographic protocols, consensus design, succinct arguments and folding schemes, elliptic curve-based STARK protocols, post-quantum security and zero-knowledge proofs (ZKPs). Our research has led to influential contributions, including Zinc (Crypto '25), Mova, FLI (Asiacrypt '24), and foundational results in Fiat-Shamir security and STARK proof batching. Complementing this theoretical work, our engineering expertise is demonstrated through implementations such as the Latticefold aggregation scheme, the Labrador proof system, zkvm-benchmarks, and Plonk Verifier in Cairo. This combined strength in theory and engineering enables us to deliver cutting-edge cryptographic solutions to partners and clients.

Smart Contract Development & DeFi Research: Our smart contract development and DeFi research team comprises 40+ world-class engineers who collaborate closely with partners to identify needs and work on value-adding projects. The team specializes in Solidity and Cairo development, architecture design, and DeFi solutions, including DEXs, AMMs, structured products, derivatives, and money market protocols, as well as ERC20, 721, and 1155 token design. Our research and data analytics focuses on three key areas: technical due diligence, market research, and DeFi research. Utilizing a data-driven approach, we offer in-depth insights and outlooks on various industry themes.

Our suite of L2 tooling: Warp is Starknet's approach to EVM compatibility. It allows developers to take their Solidity smart contracts and transpile them to Cairo, Starknet's smart contract language. In the short time since its inception, the project has accomplished many achievements, including successfully transpiling Uniswap v3 onto Starknet using Warp.

- **Voyager** is a user-friendly Starknet block explorer that offers comprehensive insights into the Starknet network. With its intuitive interface and powerful features, Voyager allows users to easily search for and examine transactions, addresses, and contract details. As an essential tool for navigating the Starknet ecosystem, Voyager is the go-to solution for users seeking in-depth information and analysis;
- **Horus** is an open-source formal verification tool for StarkNet smart contracts. It simplifies the process of formally verifying Starknet smart contracts, allowing developers to express various assertions about the behavior of their code using a simple assertion language;
- **Juno** is a full-node client implementation for Starknet, drawing on the expertise gained from developing the Nethermind Client. Written in Golang and open-sourced from the outset, Juno verifies the validity of the data received from Starknet by comparing it to proofs retrieved from Ethereum, thus maintaining the integrity and security of the entire ecosystem.

General Advisory to Clients

As auditors, we recommend that any changes or updates made to the audited codebase undergo a re-audit or security review to address potential vulnerabilities or risks introduced by the modifications. By conducting a re-audit or security review of the modified codebase, you can significantly enhance the overall security of your system and reduce the likelihood of exploitation. However, we do not possess the authority or right to impose obligations or restrictions on our clients regarding codebase updates, modifications, or subsequent audits. Accordingly, the decision to seek a re-audit or security review lies solely with you.

Disclaimer

This report is based on the scope of materials and documentation provided by you to Nethermind in order that Nethermind could conduct the security review outlined in **1. Executive Summary** and **2. Audited Files**. The results set out in this report may not be complete nor inclusive of all vulnerabilities. Nethermind has provided the review and this report on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. Blockchain technology remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. This report does not indicate the endorsement of any particular project or team, nor guarantee its security. No third party should rely on this report in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. To the fullest extent permitted by law, Nethermind disclaims any liability in connection with this report, its content, and any related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. Nethermind does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and Nethermind will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.