Voting Contract

Description:

The following contract is responsible for conducting polls to make deterministic decisions as to what actions need to be taken for the audit's validation.
The structure of the voting contract is such that, one can simply deploy their own voting contract for arbitration hence providing full decentralization and choice since the backend is responsible for picking the random arbiters in a poll.
The whole flow of the voting contract resides in these steps:

1. Admin creates a new poll where the audit ID and voters will be chosen.
2. The audit ID in the escrow contract must have this voting contract's address as their *arbiterprovider* for the poll to correctly work.
3. There is a time set during each poll creation that ensures that the admin can not intervene before a particular amount of time.
4. The voters then have a certain amount of time for their voting, when they can cast their votes.
5. Votes are of four kinds,
    1. NoDiscrepancies
    2. MinorDiscrepancies
    3. ModerateDiscrepancies
    4. Reject
6. Corresponding to the first three kinds of votes, are an extension to the deadline and a haircut. These are variable and can be changed from their initialized values of 7, 15 days, and 5, 15 percent to anything that the admin sees fit in the future.
7. If any of the selected voters choose to reject the audit report, that decision will trump other choices, and instantly the choice will be cast, and the vote function will perform a cross-contract call to the escrow contract and call assess_audit with "false" telling it to reject the report.
8. The vote function will calculate if it is the last vote, and if it is, it will perform the cross-contract call, otherwise, it will just save the average of all the options to send eventually.
9. Say, we have selected 5 arbiters for an audit's payment arbitration, four voters have cast their vote, and the fifth voter is a no-show. Then, the

admin/backend can perform a *force_vote* where the aggregated choice will be pushed to the escrow contract. Naturally, it can not be a *Reject* call.

10. After successful arbitration, the admin will disperse the funds to the arbiters by *release_treasury* function.

Data Storage Of Voting Contract:

1. **Arbiter [Struct]:** Stores the necessary data about the arbiter, i.e., the arbiter's address and whether they've cast their vote or not.
2. **VoteInfo [Struct]:** Stores all the information about the poll like what the audit id is, the vector of arbiters, whether the poll is active or not, available votes, decided_deadline, decided+haircut, and the admin_hit_time, the time before which admin can't force_vote.
3. **AuditArbitrationResult [Enum]:** The options of votes an arbiter can cast, that are,
   NoDiscrepancies,
   MinorDiscrepancies,
   ModerateDiscrepancies,
   Reject.
4. **Vote_id_to_info [Mapping]:** a mapping from voteId to VoteInfo.

Events of Voting Contract:

1. PollCreated: emitted when a new poll gets created by the admin
2. ArbiterVoted: Emitted when an arbiter casts their vote.
3. NoOneVotedTransferredToAdmin: In case no arbiter had voted and the admin had to force_vote hence all the arbitersshare goes to the admin, this event will be emitted.
4. FinalVotePushed: Whether it is the fifth/last vote it is a reject choice, or the admin has hit force_vote, this event will be emitted when the cross-contract call succeeds to the escrow for a particular poll.

Functions of Voting Contract:

Read Functions:

1. **get_current_vote_id**: Gives a u32 response that represents the total number of votes till now, this internally is also used to create the next poll.
2. **know_your_escrow**: A voting contract only serves a single escrow contract, and this function returns the address of the escrow contract.
3. **get_poll_info**: A public function that gives the poll info struct telling about the state of votes and whether it is active or inactive.
4. **get_time_extension_info**: If called with true, it returns minor_discrepancies' time extension, otherwise moderate.
5. **get_haircut_info**: If called with true, returns MinorDiscrepancies's haircut, otherwise returns ModerateDiscrepancies's haircut.
6. **know_arbiters_share**: returns the Balance value that will be passed in escrow for arbiters share
7. **know_your_admin**: returns the address of the administrator

Write Functions:

1. **create_new_poll**: This function can only be successfully called by the admin, it creates a new poll with information audit_id, arbiters' list, and admin_hit_time.
   Arguments: audit_id (u32), buffer_for_admin (Timestamp), arbiters (Vec<Arbiter>)
   Returns: Result<()>
2. **vote**: to be called for a particular vote_id, only addresses that are in the arbiters list of that vote ID's VoteInfo can call this function successfully. The function first checks if it is the last of the poll, if it is then all permutations will result in the cross-contract call to the escrow. If the arbiter chooses to reject, or NoDiscrepancies (and all other voters had also chosen NoDiscrepancies) then assess_audit will be called false, and true respectively.
   In other cases, the average deadline extension and haircut will be passed to escrow's arbiters_extend_deadline function.
   If it is not the final vote, then if the vote option is *Reject* it will result in a cross-contract call, otherwise only the state of voteInfo will be changed.
   Arguments: vote_id (u32), result (AuditArbitrationResult).
   Returns: Result<()>

3. **force_vote**: This function can only be called by the admin on the active polls.
   The function will force the vote to the escrow and deactivate the poll. Naturally, this will only result in the timeline being extended, or the assessment of the audit with “true”.
   Arguments: vote_id (u32)
   Returns: Result<()>
4. **release_treasury_funds**: The function is to disperse the arbitration reward among the arbiters who cast their vote.
   95% of the reward will be divided amongst the arbiters, and 5% will be kept for the admin/backend hosting.
   This takes the amount manually.
   Arguments: vote_id(u32), amount (Balance).
   Returns: Result<()>
5. **flush_out_tokens**: This function can be used by the admin to take out their part of the rewards and also take out any other tokens that have been sent accidentally to this address.
   Arguments: token_address (AccountId), value (Balance).
   Returns: Result<()>
6. **change_haircut_for_discrepancies**: This can only be called by the admin to change the haircut for minor or major discrepancies.
   Arguments: change_minor(bool), new_haircut(Balance).
   Returns: Result<()>
7. **change_time_extension_for_discrepancies**: This can only be called by the admin to change the time_extension for minor or major discrepancies.
   Arguments: change_minor(bool), new_extension(Balance).
   Returns: Result<()>
8. **change_arbiters_share**: This function can only be called by the admin to change the percentage of arbiters_share.
   Arguments: new_share(Balance)
   Returns: Result<()>