Escrow Contract

Description :

The following contract will serve as an escrow contract, it is designed in a way that it is independent of any users/administrators. Anyone can use this escrow contract for their business on the chain.
The escrow contract manages the status of every audit, and transfers or allows the transfers of locked tokens only during valid states.

Data storage of escrow contract:

1. **Audit Status [enum]**: stores the status of the audit, e.g. whether it has just been created, assigned, submitted, is awaiting validation, completed, or expired.
2. **Payment Info [struct]**: The payment info struct stores all the important information related to a particular audit. It stores the patron's, auditor's, and arbiter provider's account ID. It also stores the value locked, deadline, start time, and the current status of the audit.
3. **IncreaseRequest [struct]**: The structure stores the haircut percentage the auditor is willing to take on the value, and new deadline that s/he is proposing.
4. **auditID -> PaymentInfo [mapping]**: this mapping stores payment info details of each audit.
5. **auditID -> time_increase_request [mapping]**: maps out the timeIncreaseRequest for each auditID if any.
6. **auditID -> ipfs_hash [mapping]**: mapping stores the final ipfs hash of completed audits eliminating the need for an extra storage contract.

Events of escrow contract:

1. AuditIDAssigned: emitted when an audit ID is assigned to an auditor.
2. AuditInfoUpdated: Emitted when the payment_info of for an audit ID is updated.
3. DeadlineExtendRequest: emitted when an auditor requests additional time, mainly to inform the patron and the backend.
4. AuditSubmitted: emitted when auti is submitted, so that the ipfs files can be fetched via the backend and the patron/arbiter provider.
5. TokenIncoming: When tokens are locked into the escrow contract for an auditID.
6. TokenOutgoing: When tokens are released from the escrow, maybe as haircut, or completion value, or after the expiration of the audit.
7. AuditIdRetrieved: emits and informs the retrieval of the audit ID.

Functions of escrow contract:

Read functions:

1. **get_current_audit_id**: This function allows everyone to peek at the current audit ID.
   Returns: the current audit ID (u32).

2. **know_your_stablecoin**: This function returns the account ID of the stablecoin used in the escrow contract.

3. **get_payment_info**: This function returns the payment info of the requested audit ID.
   arguments: Audit ID (u32)
   returns: PaymentInfo (option/struct)

4. **query_timeincreaserequest**: this function returns the info regarding the time increase request for a given audit ID.
   arguments: Audit ID (u32)
   returns: IncreaseRequest (Option/struct)

Write functions

1. **create_new_payment**: this function is used by a patron to create the details of a new audit, since all the details are not available at the time of the audit creation, some values are only supposed to be assigned/changed later when the audit is assigned to an auditor.

   Arguments:
   value (balance), arbiter_provider (AccountID), deadline (u64).

   Returns: Result<>

2. **assign_audit**: this functions is meant to be called only by the patron of the auditID, and it assigns an auditor, and updates the status of the audit from created to assigned.
   Arguments: id (u32), auditor (Account ID)
   Returns : Result<>

3. **request_additoinal_time**: This function is for the auditors assigned to the audits, where they can request additional time in exchange for a haircut to the original value of the audit.
   Arguments: id (u32), _time(u64), _haircut_percentage (balance)
   returns: Option<>

4. **approve_additional_time**: To be called by the patron after the auditor has requested for additional time, the function checks if the patron is calling, and is by default the approval for additional time, other option is to simply ignore the request. The function transfers the haircut value to the patron.
   arguments: id(u32)

returns: Result<>

5. **mark_submitted**: the function to be called by the auditors to update the status of the project, and upload the ipfs hash of the audit report/files.
   Arguments: id (u32), _ipfs_hash (String)
   returns: Result<>

6. **assess_audit**: The function has multiple routes, it can be called by the patron or the arbiterprovider, if the patron calls, and is satisfied with the audit results, the audit is then completed, the value transferred to the auditor.
   If the patron is not satisfied, the event is fired for arbiter_providers to take a look into the matter, and they will then deal with the situation, and not the patron.

   If the arbiterprovider calls, they are basically here to say yes, and make the value transfer.
   If the arbiter provider is not satisfied by the results, there are different functions to deal with the situation.
   arguments: id (u32), answer (bool)
   returns: Result<>

7. **arbiters_extend_deadline**: If the arbiters have suggested extending the deadline and made a suitable, reasonable haircut, the arbiterproviders will hit this function for that.
   Arguments:
   id (u32), new_deadline (u64), haircut (Balance), arbitersshare (Balance).
   returns: Result<>

8. **Retrieve_audit**: Function to be called by the patron in cases of mishap.
   arguments: id(u32)
   returns: Result<>