**CRYSTAL ACHIENG** **DATE: 13/05/2022**

**Course: CP108 PLUTUS/HASKELL 1**

**Problem statement:**

Most African governments are trying different measures to eliminate corruption, bad governance, mismanagement and lack of accountability in their countries but these efforts are mostly unsuccessful. As a tool for change, blockchain can help in solving some of these governance issues plaguing Africa.

**Task:**

With this in mind, think of a way you can implement a smart contract that can help eliminate any of these ills, then implement it using Plutus.

**Project:**

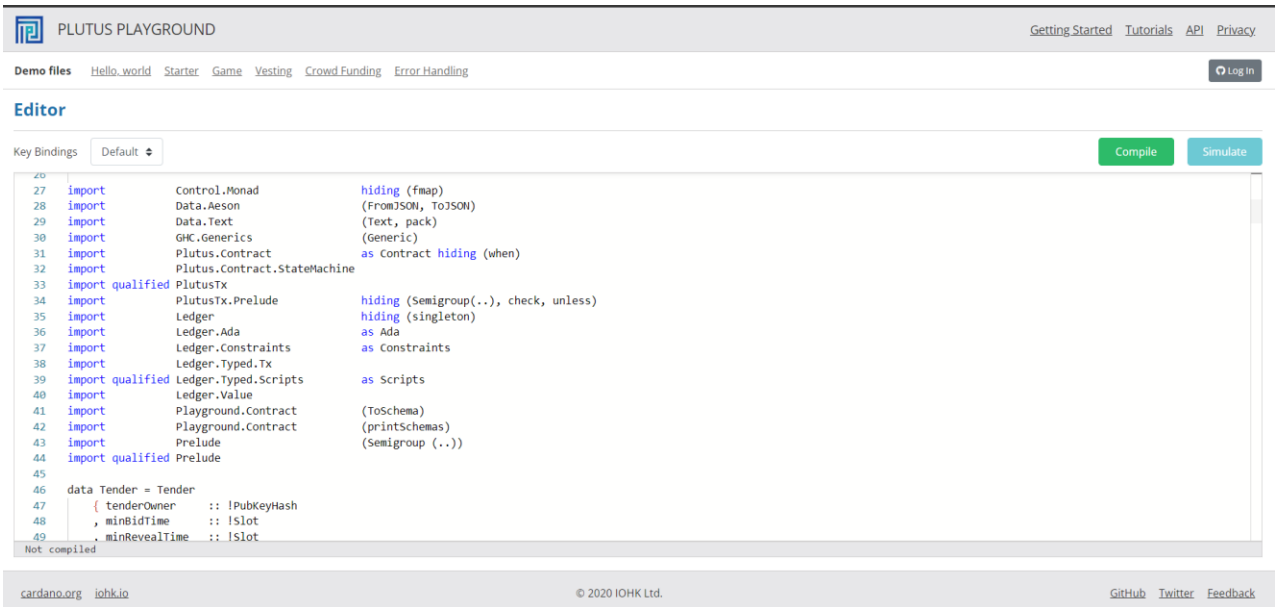
A smart contract on the Cardano blockchain that can help eliminate corruption and/or bad governance in Africa.

**Suggested solution:**

Introduce blockchain voting systems. A way that ensure that elections are done in a transparent, tamper proof and fair conduct.

The goal of this project is to create a smart contract on the Cardano blockchain that could be used as voting systems by electoral bodies. Through this we ensure secure, transparent, and above all fair elections in African countries and also all around the globe.

**Script**: the **Fabio Thomaz Molinar_Tender.hs** can be used for this exam and can be tested on the Plutus Playground. https://playground.plutus.iohkdev.io/



The smart contract is meant to work on a base that no one including its operators can see the votes, which will be taken inform of bids, already put in ballot until the election process is over. This will help keep human interference to a minimum and ensure that whatever the results of the elections are, they are true and fair.

Here is a direct link to the code to be used herein:

[https://github.com/TechEBU/Fabio-Thomaz-Molinar-Project/blob/main/Fabio%20Thomaz%20Molinar_Tender.hs](https://github.com/TechEBU/Fabio-Thomaz-Molinar-Project/blob/main/Fabio%20Thomaz%20Molinar_Tender.hs)

and also an example of the same:

324 lines (282 sloc) | 13.9 KB

Raw Blame

```haskell
{-# LANGUAGE DataKinds             #-}
{-# LANGUAGE DeriveAnyClass        #-}
{-# LANGUAGE DeriveGeneric         #-}
{-# LANGUAGE FlexibleContexts      #-}
{-# LANGUAGE MultiParamTypeClasses #-}
{-# LANGUAGE NoImplicitPrelude     #-}
{-# LANGUAGE OverloadedStrings     #-}
{-# LANGUAGE ScopedTypeVariables   #-}
{-# LANGUAGE TemplateHaskell       #-}
{-# LANGUAGE TypeApplications      #-}
{-# LANGUAGE TypeFamilies          #-}
{-# LANGUAGE TypeOperators         #-}

-- TO DO: expose objects

module Tender
    ( Tender       (..)
    , CreateParams (..)
    , BidParams    (..)
    , RevealParams (..)
    , CancelParams (..)
    , FinishParams (..)
    , TenderSchema
    , endpoints
    ) where

import           Control.Monad                hiding (fmap)
import           Data.Aeson                   (FromJSON, ToJSON)
import           Data.Text                    (Text, pack)
import           GHC.Generics                 (Generic)
import           Plutus.Contract              as Contract hiding (when)
import           Plutus.Contract.StateMachine
import qualified PlutusTx
import           PlutusTx.Prelude             hiding (Semigroup(..), check, unless)
import           Ledger                       hiding (singleton)
import           Ledger.Ada                   as Ada
import           Ledger.Constraints           as Constraints
import           Ledger.Typed.Tx
import qualified Ledger.Typed.Scripts         as Scripts
import           Ledger.Value
import           Playground.Contract          (ToSchema)
import           Prelude                      (Semigroup (..))
import qualified Prelude

```

```haskell

data Tender = Tender
    { tenderOwner     :: !PubKeyHash
    , minBidTime      :: !Slot
    , minRevealTime   :: !Slot
    , txCost          :: !Integer
    , tToken          :: !AssetClass
    } deriving (Show, Generic, FromJSON, ToJSON, Prelude.Eq, Prelude.Ord)

PlutusTx.makeLift ''Tender

data BidInfo = BidInfo
    { bidOwner     :: !PubKeyHash
    , bidValue     :: !Integer
    } deriving (Show, Generic, FromJSON, ToJSON, ToSchema, Prelude.Eq, Prelude.Ord)

-- Need to defined an instance of Eq only because I really need an instance of Ord
instance Eq BidInfo where
    {-# INLINABLE (==) #-}
    BidInfo i == BidInfo i' = bidValue i == bidValue i'

-- Defining an instance of Ord to make it easier to sort bid infos
instance Ord BidInfo where
    {-# INLINABLE (<=) #-}
    BidInfo i <= BidInfo i' = bidValue i <= bidValue i'

PlutusTx.unstableMakeIsData ''BidInfo

data TenderDatum = TenderDatum (Maybe [ByteString]) (Maybe [BidInfo]) | Finished
    deriving Show

instance Eq TenderDatum where
    {-# INLINABLE (==) #-}
    TenderDatum i == TenderDatum i' = bidValue i == bidValue i'

PlutusTx.unstableMakeIsData ''TenderDatum

data TenderRedeemer = Create
                    | Bid ByteString
                    | Close
                    | Reveal ByteString BidInfo
                    | Finish
                    | Cancel
                    deriving Show

PlutusTx.unstableMakeIsData ''TenderRedeemer

```

```
  where
    token :: Value
    token = assetClassValue (tToken tender) 1

{-# INLINABLE final #-}
final :: TenderDatum -> Bool
final Finished = True
final _        = False

{-# INLINABLE bidInfoToBS #-}
bidInfoToBS :: BidInfo -> ByteString
bidInfoToBS bi = sha2_256 (appendByteString (getPubKeyHash $ bidOwner bi) (consByteString $ bidValue bi))

{-# INLINABLE check #-}
check :: ByteString -> TenderDatum -> TenderRedeemer -> Bool
check bs (TenderDatum (Just bs) _) (Reveal bs' bi') = bidInfoToBS bi' == bs

{-# INLINABLE tenderStateMachine #-}
tenderStateMachine :: Tender -> ByteString -> StateMachine TenderDatum TenderRedeemer
tenderStateMachine tender bs = StateMachine
    { smTransition  = transition tender
    , smFinal       = final
    , smCheck       = check bs
    , smThreadToken = Just $ tToken tender}

{-# INLINABLE mkTenderValidator #-}
mkTenderValidator :: Tender -> ByteString -> TenderDatum -> TenderRedeemer -> ScriptContext -> Bool
mkTenderValidator tender bs = mkValidator $ tenderStateMachine tender bs

type Tendering = StateMachine TenderDatum TenderRedeemer

tenderStateMachine' :: Tender -> StateMachine TenderDatum TenderRedeemer
tenderStateMachine' tender = tenderStateMachine tender bs

tenderInst :: Tender -> Scripts.ScriptInstance Tendering
tenderInst tender = Scripts.validator @Tendering
    ($$(PlutusTx.compile [|| mkTenderValidator ||])
        `PlutusTx.applyCode` PlutusTx.liftCode tender
        `PlutusTx.applyCode` PlutusTx.liftCode bs)
    $$(PlutusTx.compile [|| wrap ||])
  where
    wrap = Scripts.wrapValidator @TenderDatum @TenderRedeemer

tenderValidator :: Tender -> Validator
tenderValidator = Scripts.validatorScript . tenderInst

tenderAddress :: Tender -> Ledger.Address
```

With this smart contract we eliminate corruption in electoral bodies especially on presidential candidate seats. Then African countries are able to see the change in their countries with their fairly elected leaders. This will enhance including the economic and general development in African countries.