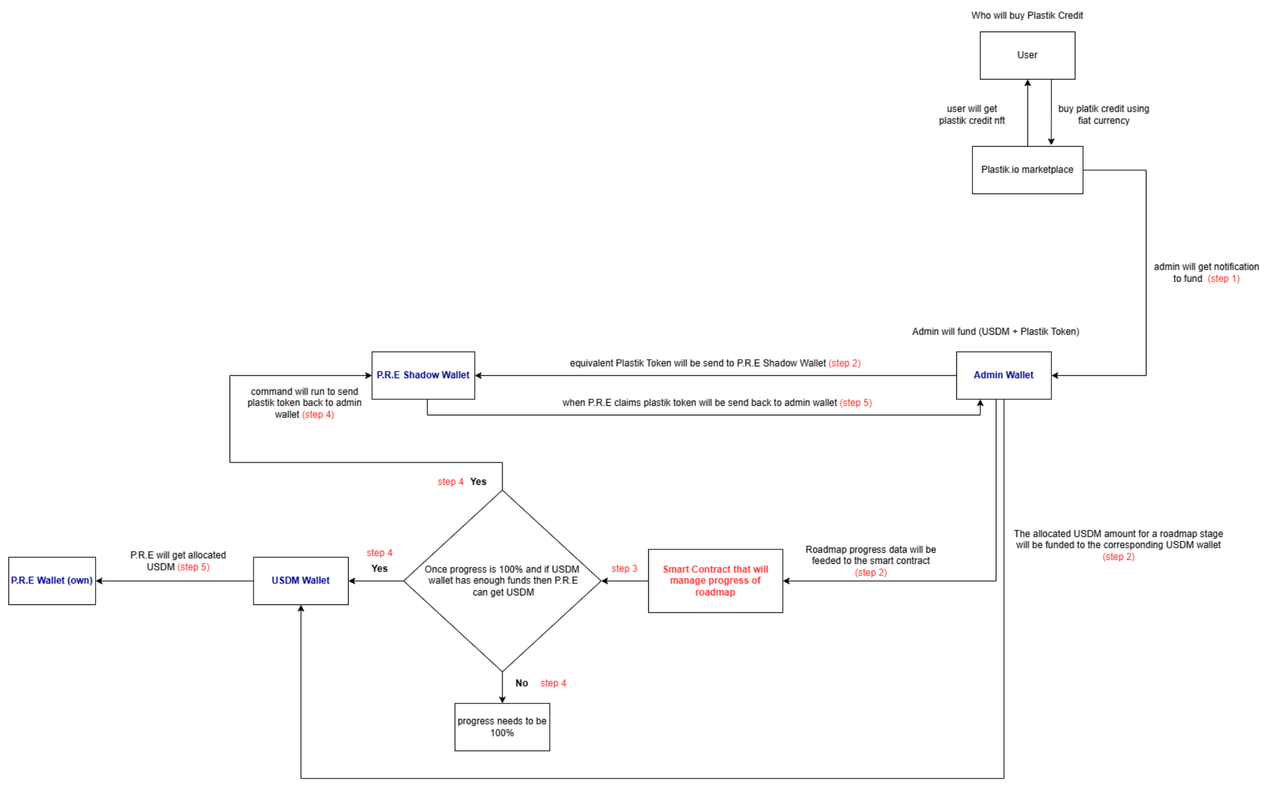
Plastiks Smart Contract Technical Documentation

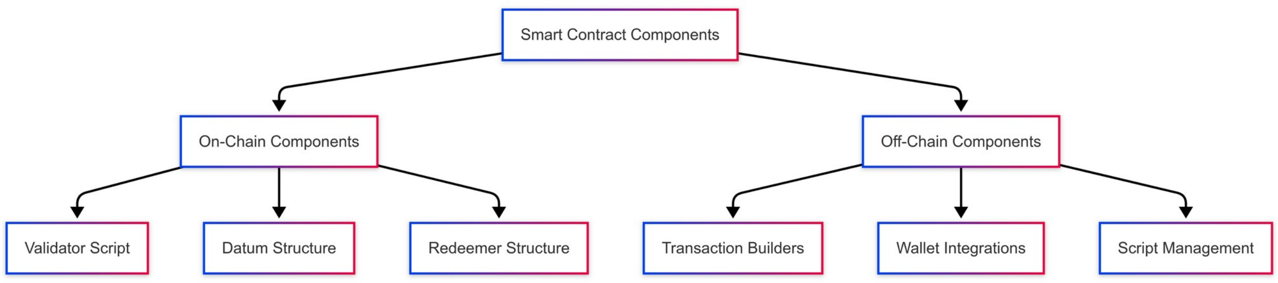
Table of Contents

1. Architectural Overview


Who will buy Plastik Credit
User
user will get plastik credit nft
buy platik credit using fiat currency
Plastik.io marketplace
admin will get notification to fund (step 1)
Admin will fund (USDM + Plastik Token)
Admin Wallet
equivalent Plastik Token will be send to P.R.E Shadow Wallet (step 2)
P.R.E Shadow Wallet
when P.R.E claims plastik token will be send back to admin wallet (step 5)
command will run to send plastik token back to admin wallet (step 4)
step 4 Yes
The allocated USDM amount for a roadmap stage will be funded to the corresponding USDM wallet (step 2)
Roadmap progress data will be feeded to the smart contract (step 2)
Smart Contract that will manage progress of roadmap
step 3
Once progress is 100% and if USDM wallet has enough funds then P.R.E can get USDM
step 4
Yes
USDM Wallet
P.R.E will get allocated USDM (step 5)
P.R.E Wallet (own)
No step 4
progress needs to be 100%

1.1 Smart Contract Components

Component Hierarchy:



1.2 Core Data Structures

PlastiksDatum (State Representation)

```
data PlastiksDatum = PlastiksDatum
  { preId :: BuiltinByteString        -- Unique project identifier
  , roadmapId :: BuiltinByteString    -- Roadmap reference
  , progress :: Integer               -- Completion percentage (0-100)
  , adminPkh :: PubKeyHash            -- Admin authority
  , prePkh :: PubKeyHash              -- Project owner
  , totalPlasticCredits :: Integer    -- Total available credits
  , soldPlasticCredits :: Integer     -- Credits sold
  , totalPlasticTokens :: Integer     -- Total tokens minted
  , sentPlasticTokens :: Integer      -- Tokens distributed
  }
```

PlastiksRedeemer (State Transition)

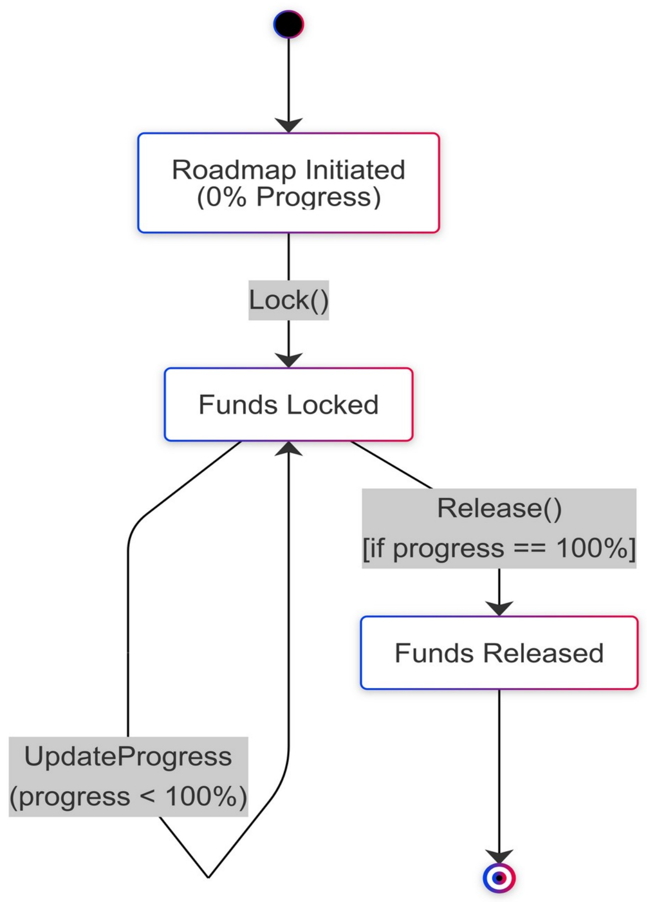
```
data PlastiksRedeemer
  = UpdateProgress Integer  -- Progress update operation
  | Release                 -- Fund release operation
```

1.3 Validation Logic

```
validate :: PlastiksDatum -> PlastiksRedeemer -> ScriptContext -> Bool
validate datum redeemer ctx =
  case redeemer of
    UpdateProgress newProgress ->
      traceIfFalse "Admin not signed" (txSignedBy info (adminPkh datum)) &&
      traceIfFalse "Invalid progress update" (newProgress > progress datum && newProgress <=
100)

    Release ->
      traceIfFalse "Admin not signed" (txSignedBy info (adminPkh datum)) &&
      traceIfFalse "Progress not complete" (progress datum == 100)
  where
    info :: TxInfo
    info = scriptContextTxInfo ctx
```

2. Transaction Lifecycle

2.1 State Transition Diagram



2.2 Key Operations

Operation	Script	Parameters	Required Signatures
Contract Lock	lock.sh	Initial datum	N/A
Progress Update	lock-update.sh	New progress value	Admin
Fund Release	unlock.sh	Release redeemer	Admin

3. Security Model

3.1 Authorization Details

Role	Update Progress	Release Funds	Modify Parameters
Admin	✓	✓	✓
Third Party	✗	✗	✗

3.2 Validation Checks

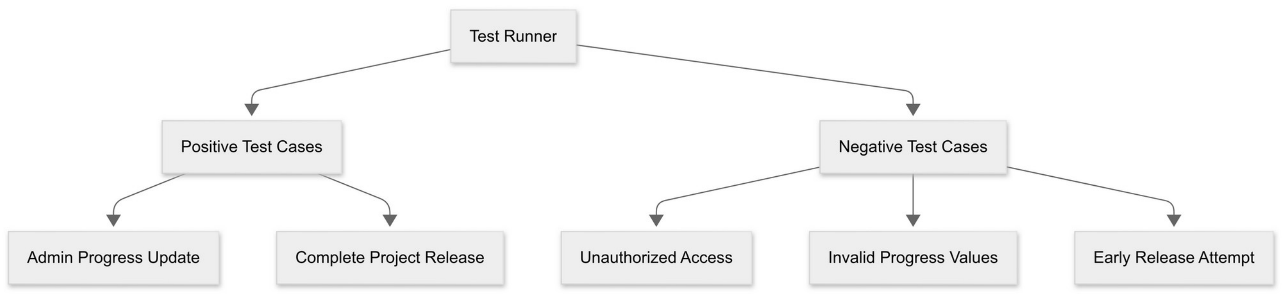
```
-- Progress update constraints
newProgress > currentProgress && newProgress ≤ 100

-- Release constraints
progress ≡ 100 && txSignedBy adminPkh

-- Universal checks
¬(totalPlasticCredits < soldPlasticCredits) &&
¬(totalPlasticTokens < sentPlasticTokens)
```

4. Testing Strategy

4.1 Test Coverage Details

Test Case	Validator Path	Expected Outcome
Valid Progress Update	UpdateProgress	Success
Unauthorized Progress Update	Missing Admin Sig	Failure
Progress Regression	Lower Value	Failure
Premature Release	Progress < 100	Failure
Valid Fund Release	Progress = 100	Success

4.2 Test Execution Flow



5. Deployment Architecture

5.1 Environment Configuration

```
network: preprod
node_version: 1.35.5
plutus_version: 2.0.0
required_tools:
  - cardano-cli
  - cardano-wallet
  - plutus-script-utils
```

5.2 Smart Contract Compilation Workflow

```
# Compilation Process
cabal build → Generate Haskell Executable
             ↓
             Compile Plutus Script → refi.json
             ↓
             Generate Script Address → refi.addr
```

6. Error Handling

6.1 Validation Errors

Error Code	Message	Resolution
VAL001	Admin not signed	Verify transaction signature
VAL002	Invalid progress update	Check progress constraints
VAL003	Progress not complete	Achieve 100% before release

6.2 Operational Errors

```
{
  "error": "INSUFFICIENT_FUNDS",
  "solution": [
    "Verify wallet balance",
    "Check UTXO selection",
    "Confirm network parameters"
  ]
}
```

7. References

1. Plinth User guide:

Official documentation covering Plutus Core, PlutusTx, and writing validators.

🔗 https://plutus.cardano.intersectmbo.org/docs/

2. Cardano Developer Portal:

A must-visit hub for all things Cardano dev — tools, APIs, smart contracts, and examples.

🔗 https://developers.cardano.org/

3. Hackage:

The Haskell Package Repository

🔗 https://hackage.haskell.org/

4. Extended UTXO Model Paper:

Understand how Cardano differs from Ethereum's account model (important for designing contracts).

🔗 https://iohk.io/en/research/library/papers/the-extended-utxo-model/

5. Real-world Plutus Contracts (e.g., Mlabs):

Contains production-grade smart contracts and patterns used in real-world DApps.

🔗 https://github.com/mlabs-haskell/plutus-use-cases

6. Plutus Pioneer Program:

IOHK's official beginner-to-advanced course with guided code examples and videos.

🔗 https://github.com/input-output-hk/plutus-pioneer-program

7. IOHK YouTube – Plutus Playlist(Plutus Pioneer Program)

🔗 https://youtube.com/playlist?list=PLnPTB0CuBOBypVDf1oGcsvnJGJg8h-LII&si=1k_r3XkFSTjhTvB5