Practical Examen BPDA

5 February 2024

- There are 60 minutes for this practical exam;
- You can access **any non-collaborative resource** including:
  - https://cs-pub-ro.github.io/blockchain-protocols-and-distributed-applications/
  - https://docs.multiversx.com/developers/smart-contracts
  - https://github.com/multiversx/mx-contracts-rs/
- You can solve the tasks on any blockchain you want.
- You can solve the tasks in any order you want.
- **Total Points 110p**; 100p will get you the maximum grade.

TEACHER_ADDR_MX= erd1ld6er5zpdze3cynzkapur9qhzh826jje6n87g7tvdfrtszs8jn2qv44nqd
TEACHER_ADDR_ETH= 0x8270685220fAd7EDb232E649EeE9D8810dac1d58

---

1. Create a wallet on testnet/devnet. You will use this wallet to solve the following tasks. **(5p)**
   a. Call Faucet to get xEgld.

2. Issue an NFT collection: **(15p)**
   a. Token Name: **BPDAExamNFT**;
   b. Token Ticker: **BPDANFT**;
   c. Based on this collection, create an NFT (10p)
      i. Set the royalties to 7%;
      ii. Any other fields are not relevant for this exam;
      iii. Send it to **$TEACHER_ADDR**;

3. Issue a Fungible Token: **(20p)**
   a. Token Name: **BPDAExamFT**;
   b. Token Ticker: **BPDA**
   c. Initial supply: 1991 tokens;
   d. Number of decimals: 18;
   e. Mint additional 9 tokens (5p)
   f. Send 500 tokens to **$TEACHER_ADDR**; (10p)

4. Write a smart contract for a **Token Marketplace** on a blockchain platform). The smart contract should allow users to list, buy, and sell fungible or non-fungible tokens. **(70p)**

   Your smart contract should meet the following specifications:
   a. **Token Listing**:
      i. Users can list their tokens (Fungible or NFTs) for sale by specifying:
         - Token type (Fungible or NFT).

        - Token identifier.
        - Quantity (for fungible tokens).
        - Price per token.
    b. **Token Purchase**:
        i. Buyers can purchase tokens by sending the specified amount in the blockchain's native currency (e.g., EGLD, ETH).
        ii. Ensure proper fund transfer and ownership update.
    c. Token Removal:
        i. Sellers can remove their token listing at any time before a purchase is made.
    d. **Events**:
        i. Emit events for the following actions:
            - Token listed for sale.
            - Token purchased.
            - Token listing removed.

Endpoint names you can use for the **Token Marketplace** smart contract.

Endpoints for Token Listing (25p)

1. **List Token for Sale**
    - **Name**: `listToken`
    - **Description**: Allows a user to list their token for sale by specifying details like token ID, quantity, and price.
    - **Inputs**:
        - `tokenId`: ID of the token.
        - `quantity`: Number of tokens to sell (for fungible tokens).
        - `price`: Price per token.
2. **Remove Token Listing**
    - **Name**: `removeListing`
    - **Description**: Allows the seller to remove their token listing.
    - **Inputs**:
        - `listingId`: ID of the token listing.
3. **Get All Listings**
    - **Name**: `getListings`
    - **Description**: Returns all active token listings.
    - **Inputs**: None.
4. **Get My Listings**
    - **Name**: `getMyListings`
    - **Description**: Returns all token listings created by the current user.
    - **Inputs**: None.

Endpoints for Token Purchase (25p)

5. **Buy Token**
    - **Name**: `buyToken`
    - **Description**: Allows a user to buy a token by specifying the listing ID and quantity (if applicable).
    - **Inputs**:
        - `listingId`: ID of the token listing.
        - `quantity`: Number of tokens to buy (for fungible tokens).
6. **Get Purchased Tokens**
    - **Name**: `getPurchasedTokens`
    - **Description**: Returns the list of tokens purchased by the user.
    - **Inputs**: None.

Events (20p)

Define corresponding **events** for transparency and notifications:

1. `TokenListed` – Emitted when a token is listed for sale.
2. `TokenPurchased` – Emitted when a token is successfully purchased.
3. `ListingRemoved` – Emitted when a token listing is removed.
4. `AuctionCreated` – Emitted when an auction is created.
5. `BidPlaced` – Emitted when a bid is placed.
6. `AuctionEnded` – Emitted when an auction ends successfully.