Individual assignment

In this individual coursework, you have to complete the following two exercises. You can achieve a maximum of **50 points** in each of them.

1. Smart contract for marketplaces in DAML

In this first exercise you have to build in a smart contract template in DAML that can be used for decentralized marketplaces. To do this, you are asked to create a new contract template called `Itemtosell` that the `seller` can use to advertise a certain object and transfer ownership to a `buyer`. We also wish to add an `insurer` that can only view the information about the trade (without being able to issue or modify the contract `Itemtosell`).

Contract template

1. Start by creating a new project and .daml file.
2. Create a contract template called `Itemtosell`. [8 pts] In the contract you must specify the following:

- Seller - Owner - Insurer - Type of item - Quantity - Price - Currency - Country

3. Then, define the roles of the parties. What type of party should the seller be? And the insurer? [8 pts]
4. Add conditions to make sure that (i) the quantity and (ii) the price are larger than zero. [5 pts]
5. Add a function `ChangePrice` to modify the price listed in the contract where the controller is the “owner” of the item. [5 pts]
6. Add a function `Sell` that modifies the owner of the item listed in the contract where the controller is the “owner” of the item. [5 pts]

Scenario testing

In the scenario testing part, using the following structure

``` setup = scenario do

-- Add parties

--Create contract

-- Trasnfer ownership

return() ```

you must:

1. Create three parties: “Party 1” (the seller), “Party 2” (the buyer), “Party 3” (the insurer). [2 pts]
2. Let the seller “Party 1” issue a new contract where the seller “Party 1” wishes to sell 1 watch for 100 GBP to “Party 2”. At this stage set the owner = the seller [5 pts]
3. Let the owner (=seller) of the watch transfer the ownership to “Party 2” (the buyer). [5 pts]
4. Take a screenshot of the state of the ledger (by looking at the scenario results). [2 pts]
5. Describe one example where the execution of the contract would fail automatically and explain why this would happen. [5 pts]

### Deliverable

Upload one PDF file that includes the following information: - Complete smart contract code; - Provide answers and explanations requested above.

## 2. Create an `assignmentToken`

In this second exercise, you are expected to create and deploy an ERC-20 token.

### Instructions

#### Coding smart contract

Create a token with the following features:

1. Mintability
   - The token can be minted (increasing supply)
   - Only a `minter` can mint the token
   - The original `minter` is the contract creator
   - A `minter` can transfer "mintership" to another address
2. Burnability
   - The token can be burned (reducing supply)
3. Capped supply
   - The initial supply of the token at contract creation is 50,000, which is credited to the contract creator's balance
   - The total supply of the token is capped at 1,000,000
4. Token transfer fee
   - A flat fee of 1 unit of the token is levied and rewarded to the `minter` with every transfer transaction (`mint` or `burn` not included)
   - A transfer transaction must be able to cover the transaction fee in order to succeed

You can find the skeleton code here. You are recommended to use Remix for composing the smart contract.
```

Deploying smart contract

Deploy the completed smart contract on Kovan or Ropsten testnet and verify the contract code on Etherscan.

Interacting with smart contract

Interact with the deployed smart contract by performing the following 5 transactions:

1. mint 60 new tokens to an address (say, address `XYZ`) other than the minter (say, address `ABC`)
2. burn 70 tokens from address `ABC`
3. approve address `XYZ` to spend up to 110 tokens from address `ABC`
4. transfer mintership to address `XYZ`
5. transfer 40 tokens with address `XYZ` from address `ABC` to a third address

Deliverable

Upload one PDF file that includes the following information:

1. Complete smart contract code: **28 points**

- e.g.

```
 contract MyToken {
// total supply of token
uint256 constant supply = 1000000;

function allowance(address _owner, address _spender) public view returns (uint256 remaining) {
remaining = allowances[_owner][_spender];
return remaining;
}
}
```

1. Deployed smart contract url: **7 points**
   - e.g. https://kovan.etherscan.io/address/0x714adeedb372ce1307d69cca1dfc694a4ec587ed#code
2. Transaction urls (one url per transaction): **15 points**
   - e.g. transfer tokens: https://kovan.etherscan.io/tx/0x8e70f74b846f200b43ad27e10bd3bea9ef741be90f73b300fc24abaed22fc25e

Deadline

You must upload your PDF onto Moodle by **16:00 (UK time) on Wednesday 17 November 2021**.