School of Computing: assessment brief

Module title	Blockchain Technologies
Module code	COMP5125M
Assignment title	Coursework
Assignment type and description	It is a programming assignment where students are required to develop smart contracts for peer-to-peer energy trading.
Rationale	The aim of this assignment is to evaluate the students' knowledge of blockchain based systems and smart contract development skills
Word limit and guidance	N/A
Weighting	60%
Submission deadline	02/05/2023
Submission method	Student will upload their assignment on Gradescope
Feedback provision	Feedback will be provided online through Minerva and Gradescope
Learning outcomes assessed	Design decentralized applications and implement smart contracts, Understand the context to which distributed ledgers are applied
Module lead	Evangelos Pournaras
Other Staff contact	Rabiya Khalid (r.khalid@leeds.ac.uk)

1. **Assignment guidance**

   Consider a smart community of electricity prosumers (electricity consumers who also produce electricity). There can be a mismatch between energy demand of a prosumer and the produced energy during a specific time interval. When a prosumer produces more energy than its requirement, it can sell it to the other prosumers of the community who do not have enough energy to fulfill their need. Traditionally, a third party manages the trading of electricity between prosumers; however, a significant cost is paid to the third party for its services and the system is prone to security and privacy issues. To mitigate these challenges, a blockchain based energy trading mechanism emerged as a promising solution. In this coursework, you are required to design a smart contract for peer-to-peer (P2P) energy trading between electricity prosumers.

   Develop a main smart contract for energy trading for a local energy market where prosumers send requests for buying and selling energy. The smart contract should match the energy requests of buyers with sellers and trade the energy. In the smart contract, the prosumers must register before taking part in energy trading. When a new prosumer is registered on the network, a smart wallet for the prosumer is automatically created by the smart contract. To purchase the energy, a prosumer must have Ethers in his smart wallet (user can send ethers to smart contract). A registered prosumer should be able to send energy surplus (sell) or deficit (buy) request to the main smart contract: positive value for surplus energy and negative value for deficit energy. On getting the request, the main smart contract should check if the requester is buyer or seller and call the respective energy trading function of P2P smart contract (in case of buyer, main smart contract also checks if the buyer has sufficient balance in his smart wallet to purchase the required amount of energy, for instance, assume for 1 unit of energy, a buyer needs to pay 1 Ether).

   On getting the energy buying request, the P2P smart contract should first find the available energy seller in the market who has sufficient surplus energy to fulfill the energy requirement of the buyer. In case no seller is available, the buyer's request should be saved. Similarly, on getting the energy selling request, the P2P smart contract should first find the energy buyer in the market whose energy demand can be fulfilled by the seller; otherwise, the seller‘s request should be saved. After energy trading, the information should be updated in the blockchain (records of both buyer and seller should be updated), and payment of energy should be added to the energy seller's account (smart wallet). For each prosumer, its ID (address), energy status (how much energy it needs to buy or sell), and balance (Ethers in smart wallet) must be stored in the blockchain as a record and it must be updated after the prosumer takes part in energy trading. A prosumer should be able to withdraw his Ethers from smart wallet.

Traditionally, the electricity prosumers prefer to store the surplus energy in the local energy storage system (ESS) instead of selling it to the buyers. They use this energy instead of buying energy from other sellers when they are in energy deficit. Propose (and implement in the smart contract) an incentive mechanism for the energy sellers and buyers to encourage them to take part in energy trading. The price for 1 energy unit = 1 Ether but you can also implement variable pricing or a different model that depends on your incentive mechanism.

Note:

- You can use any version of the solidity, however, newer versions are considered better.
- The Main and P2P smart contracts can be two independent smart contracts or they can have parent-child relationship. Moreover, both smart contracts can be in two separate files or they can be in the same file.
- In the text document, there should be three sections description of incentive mechanism, what extra things are used to improve the smart contract along with the rationale and a table of implemented scenario. Whatever you add in the smart contract, add it in the improvement section. For example, you have used queues of sellers and buyers because ... (your reason of using queues).

2. **Assessment tasks**

   The following functions must be the part of your smart contracts. (Your smart contracts are not limited to only these functions. You can add more functions as per requirement.)

   P2P smart contract

(a) A structure to store information of prosumers. The information must include a prosumer's ID (address), energy status (how much energy it needs to buy or sell), and balance (Ethers in smart wallet). You can also add more information as per requirements. Note that the data of buyers and sellers should not be stored separately, it should be stored in the same structure named "prosumer". The data related to prosumers will only be stored in P2P smart contract. No more than one struct shall be used to store information. (You can create queues for sellers and buyers (just their addresses not details). The queues will be used for the reference purpose only.)

(b) A function to register a new prosumer (add information of a new prosumer). Initially, only address of the user is added as its ID.

(c) Add functions to buy and sell energy.

Main smart contract

(a) Modifier function to make sure a prosumer is registered in the system before sending any request.

(b) Modifier function to ensure single registration of a prosumer (if an already registered prosumer request for the registration again, the function should send an error message saying the user is already registered).

(c) A modifier function to check whether a buyer has deposited sufficient funds (Ethers required to purchase the required amount of energy) to buy energy in the smart wallet.

(d) A public function to register a prosumer. A prosumer only calls this function to get registered (prosumer does not pass any value). The function checks (using the modifier function) the prior registration of the prosumers. If prosumer is already registered, the error message is generated (by the modifier function) to show that prosumer is already registered; otherwise the address of the new prosumer is sent to the P2P smart contract for storage (registration).

(e) A public function to enable a buyer to deposit some Ethers prior to energy buying request. The prosumer dose not pass any value to the function.

(f) A public function to accept prosumers' requests and check if a prosumer has sent an energy selling or buying request and pass the data to the P2P smart contract. A prosumer passes positive value if he is a seller and negative value if he is a buyer. For example, if a buyer needs 3 units of energy, he will send -3 as input. The negative sign shows that the buyer needs the energy. On the contrary, if a seller wants to 3 units of energy, it will send 3 as an input. The positive 3 shows that the user has surplus energy to sell.

(g) A public function to check the current energy status of a seller or buyer (the amount of energy a buyer wants to buy or a seller wants to sell). The function should not have any input arguments.

(h) A public function to check the balance of a prosumer. The function should not have any input arguments.

(i) A public function to withdraw the Ethers from smart wallets of prosumers. The function should not have any input arguments. (Note: funds can be withdrawn for a prosumer if his energy status is greater than or equal to zero, which means the prosumer does not need to buy energy at the moment.)

Scenario to implement
Consider a scenario of a smart community with 20 electricity prosumers with unique addresses. All of the prosumers are registered on the blockchain. Suppose 10 prosumers are energy deficit and send the energy buying request to the smart contract. For each prosumer, randomly select energy deficit status (amount of required energy) between (1-6) units. On the other hand, 10 prosumers have surplus energy and choose their energy status (the amount of energy they want to sell) between (1-8) units. Send the requests of prosumers to the smart contract randomly and observe the behaviour of your system according to the instructions given above. Create a table and add the following information of all 20 prosumers in it:

(a) Address

(b) Initial energy status before energy trading

(c) Balance in the smart wallet before energy trading

(d) Energy status after energy trading

(e) Balance in smart wallet after energy trading

3. **General guidance and study support**

   The reading list and study support materiel will be available on Minerva and Gradescope.

4. **Assessment criteria and marking process**

   The assessment criteria is provided at the end of this document in the form of rubrics. Please refer to the point no 8 of the document.

5. **Presentation and referencing**

   You are required to upload your code file of smart contracts on Gradescope along with a text (PDF) file. In the text file, you will provide a brief description of your proposed incentive mechanism (at least 1 paragraph), a table of your system's behaviour (mentioned in point 2) and improvements made in the code with rationale. Anything implemented in the smart contracts (improvement) which is not mentioned in this document should not violate the constrains of the smart contracts and should be discussed in the text file. Your text document will have three sections: Incentive mechanism, Improvements and table of system behavior.

   The quality of written English will be assessed in this work. As a minimum, you must ensure:

   - There are links between and within paragraphs although these may be ineffective at times
   - There are (at least) attempts at referencing (if your incentive mechanism is inspired from an existing work)
   - Word choice and grammar do not seriously undermine the meaning and comprehensibility of the argument
   - Word choice and grammar are generally appropriate to an academic text

6. **Submission requirements**

   It is an individual assignment (no teams). Submit your files on grade scope and names of your files should be as follows:

Name-Surname-SmartContracts.PDF

7. **Academic misconduct and plagiarism**

   Academic integrity means engaging in good academic practice. This involves essential academic skills, such as keeping track of where you find ideas and information and referencing these accurately in your work.

   By submitting this assignment you are confirming that the work is a true expression of your own work and ideas and that you have given credit to others where their work has contributed to yours.

8. **Assessment/marking criteria grid**

	Requirement and delivery	Coding standard and documentation	Runtime efficiency
>85	Excellent implementation of all functions in respective smart contracts. All instruction related to functions and smart contracts are followed. Incentive mechanism is convincing.	Excellent use of white spaces. Creatively organized work Excellent use of variables (no extra global variables, unambiguous naming). Clearly and effectively documented including descriptions of all class variables. Specific purpose noted for each function, control structure, input requirements, and output results.	Executes without errors. Excellent user prompts. Solution is efficient and easy to understand.
80-84	Excellent implementation of all functions in respective smart contracts. All instruction related to functions and smart contracts are followed. Incentive mechanism is convincing.	Excellent use of white spaces. Creatively organized work. Poor use of variables. Clearly and effectively documented including descriptions of all class variables. Specific purpose noted for each function, control structure, input requirements, and output results.	Executes without errors. User prompts contain fair amount of information. Solution is efficient and easy to understand.

75-79	Excellent implementation of all functions in respective smart contracts. All instruction related to functions and smart contracts are followed. Incentive mechanism is convincing.	Excellent use of white spaces. Creatively organized work. Poor use of variables. Clearly and effectively documented including descriptions of all class variables. Specific purpose noted for each function, control structure, input requirements, and output results.	Executes without errors. User prompts contain little information. A logical solution that is easy to follow but it is not the most efficient.
70-74	Excellent implementation of all functions in respective smart contracts. All instruction related to functions and smart contracts are followed. Incentive mechanism is convincing.	Good organization Poor use of variables (many global variables, ambiguous naming). Fairly documented. Documentation helps the reader to understand the code.	Executes without errors. User prompts contain little information. A logical solution that is easy to follow but it is not the most efficient . Code is too large.

65-69	Complete assignment. Implemented all functions. All instruction related to functions and smart contracts are followed. Incentive mechanism is proposed.	Good organization. Poor use of variables (many global variables, ambiguous naming). Fairly documented. Documentation helps the reader to understand the code.	Executes without errors. User prompts contain little information. A logical solution that is easy to follow but it is not the most efficient. Code is too large.
60-64	Complete assignment. Implemented all the functions. All instruction related to functions and smart contracts are followed.	Good organization. Poor use of variables (many global variables, ambiguous naming). Very limited or no documentation included. Documentation does not help the reader understand the code.	Executes without errors. User prompts contain little information. A difficult to understand and inefficient solution. Code is too large.
50-59	Complete assignment. Implemented all functions.	Fairly organized. Poor use of variables (many global variables, ambiguous naming). Very limited or no documentation included. Documentation does not help the reader understand the code.	Executes without errors. User prompts missing for some functions. A difficult to understand and inefficient solution Code is too large.

<50	Incomplete assignment. Not delivered on time.	Disorganized and messy. Poor use of variables (many global variables, ambiguous naming). Very limited or no documentation included. Documentation does not help the reader understand the code.	Does not execute due to errors. User prompts are misleading or non-existent. A difficult to understand and inefficient solution. Code is too large.