CS-458 – Assignment #02: Implementing a Simple Blockchain

Due: Wed June 5, 11:59pm

Objective:

The objective of this assignment is to understand the fundamental concepts of blockchain technology by implementing a basic blockchain in JavaScript (Node.js), following the guidelines provided in the Medium article "A blockchain in 200 lines of code" by Lauri Hartikka.

Resources:

A blockchain in 200 lines of code

Instructions:

- Read the Article: Begin by thoroughly reading the article. Pay close attention to the explanations for each part of the code. This will help you understand the structure and functionality of the blockchain.
- Set Up Your Environment: Ensure you have JavaScript (Node.js) installed on your personal computer. You can also connect to the CS (Computer Science) Lab machines using a remote desktop (https://www.engr.colostate.edu/ets/virtual-classroom/) and use Visual Studio Code to get started, however, it is not required.
- **Background**: Follow the steps below to implement the blockchain (most of the functionalities are there in the article, please refer to them):
  - Create the Block Class Define a Block class that will hold data, the hash of the previous block, and its hash.
  - Define the Blockchain Class Create a Blockchain class that will manage the chain of blocks. This class should include methods for adding new blocks and validating the chain.
  - Implement Proof-of-Work: Implement a simple proof-of-work algorithm to secure the blockchain. Proof-of-Work is a consensus algorithm used to secure the blockchain by requiring some work to be done (usually in the form of solving a cryptographic puzzle) before a new block can be added.
    - In the Proof-of-Work mechanism, the process works as follows:
      - Difficulty Target: The blockchain has a difficulty target, which determines how hard it is to find a valid hash. For example, a difficulty of 4 means that the hash must start with at least four zeros.
      - Mining: To add a new block to the blockchain, a miner must find a nonce (a number used only once) such that when the block's data,

including the nonce, is hashed, the resulting hash starts with the required number of zeros. This process is computationally intensive and requires trying many different nonce values.
        - Validation: Once a miner finds a valid nonce, the block is considered mined, and its hash is recorded. This hash is included in the next block, linking the blocks together.
        - Security: The Proof-of-Work mechanism ensures that adding a new block requires significant computational effort, making it difficult for anyone to alter the blockchain. Any change to a block would require re-mining all subsequent blocks, which is computationally impractical.
- Integrate the Components Combine the Block and Blockchain classes and proof-of-work mechanism.
- Create a client (none of the miners) code that sends data to be stored in the blockchain and can query the BlockChain (like listing the data in the BlockChain)
    - To test your work, create client nodes
    - Insert the following data into the Blockchain:
        - Generate some data with (name, dates) - Record to include your name and today's date (you can insert it manually or take it from the system).

- **Task Description**:
- **Task 1:** Update the code by adding the PoW and Client code. You may need to update other parts of the code, like block integrity verification.
- **Task 2:**
    - Create three Blockchain Nodes, N1, N2, and N3, where initially N1 is connected (peered with) to N2, and N2 is connected to N1, and the connected Nodes should have the same copy of the blockchain (using peers). Leave N3 unconnected to other nodes for now.
    - Create a client node that connects to one of the Blockchain nodes and add new blocks to the blockchain (for the following list of Names: [Malfoy, Harry, Dumbledore, ***your_name***]). Use the data field of the block (the data is the name + the current date)
    - The client node connects to one of the blockchain nodes N1 or N2 and sends the data to these nodes, then the selected node mines and broadcasts the new block to other nodes, and then each node verifies the integrity of the received block and updates its chain if verification is successful.

    - Finally, connect N3 with the other nodes and see if your blockchain was able to broadcast everything properly and synchronize the nodes with N3. (Include screenshots of the Blockchain content in your report for all the steps.)
- **Documentation**: Document your code thoroughly. Explain the purpose of each class and method. Add comments to clarify complex sections of the code.
- **Submission**: Submit your complete implementation along with a brief report (1-3 pages) that includes:
    - A summary of how you implemented the blockchain and how to use it.
    - Any challenges you faced and how you overcame them.
    - Screenshots of your blockchain in action.
    - Project file in a zip folder.
- **Evaluation Criteria**:
    - Proof of work and block integrity verification code: **15 points**.
    - Client code: **15 points**.
    - Running the Nodes and adding the blocks: **15 points**.
    - Documentation: **5 points**.
    - Report with screenshots: **10 points**.