**Blockchain Technology Lab**

**Lab – 4**

**Aim :** Implement PoW Consensus Mechanism on your own Blockchain

**Code:**

*import* hashlib

*import* json

*import* os

*class* Student:

*def* \_\_init\_\_(self, name, rollnumber, branch):

*self*.name = name

*self*.rollnumber = rollnumber

*self*.branch = branch

*def* to\_dict(self):

*return* {

            "name": *self*.name,

            "rollnumber": *self*.rollnumber,

            "branch": *self*.branch

        }

    @classmethod

*def* from\_dict(cls, data):

*return* *cls*(data["name"], data["rollnumber"], data["branch"])

*class* Block:

*def* \_\_init\_\_(self, block\_id, nonce, student, previous\_hash):

*self*.block\_id = block\_id

*self*.nonce = nonce

*self*.data = student.to\_dict()

*self*.previous\_hash = previous\_hash

*self*.block\_hash = *self*.calculate\_hash()

*def* calculate\_hash(self):

        block\_dict = {

            "block\_id": *self*.block\_id,

            "nonce": *self*.nonce,

            "data": *self*.data,

            "previous\_hash": *self*.previous\_hash

        }

        block\_string = json.dumps(block\_dict, sort\_keys=True).encode()

*return* hashlib.sha256(block\_string).hexdigest()

*def* to\_dict(self):

*return* {

            "block\_id": *self*.block\_id,

            "nonce": *self*.nonce,

            "data": *self*.data,

            "previous\_hash": *self*.previous\_hash,

            "block\_hash": *self*.block\_hash

        }

    @classmethod

*def* from\_dict(cls, block\_data):

        student\_data = Student.from\_dict(block\_data["data"])

        block = *cls*(

            block\_id=block\_data["block\_id"],

            nonce=block\_data["nonce"],

            student=student\_data,

            previous\_hash=block\_data["previous\_hash"]

        )

        block.block\_hash = block\_data["block\_hash"]

*return* block

*class* Blockchain:

*def* \_\_init\_\_(self, difficulty, filename="blockchain.json"):

*self*.chain = []

*self*.difficulty = difficulty

*self*.filename = filename

*if* os.path.exists(*self*.filename):

*self*.load\_chain()

*else*:

*self*.create\_genesis\_block()

*def* create\_genesis\_block(self):

        genesis\_block = Block(

            block\_id=1, nonce=0, student=Student("Genesis", "0000", "Genesis"), previous\_hash='0')

*self*.chain.append(genesis\_block)

*self*.save\_chain()

*def* add\_block(self, student):

        previous\_block = *self*.chain[-1]

        nonce = *self*.proof\_of\_work(student, previous\_block.block\_hash)

        new\_block = Block(len(*self*.chain) + 1, nonce,

                          student, previous\_block.block\_hash)

*self*.chain.append(new\_block)

*self*.save\_chain()

*return* new\_block

*def* proof\_of\_work(self, student, previous\_hash):

        nonce = 0

*while* True:

            block\_candidate = Block(

                len(*self*.chain) + 1, nonce, student, previous\_hash)

            block\_hash = block\_candidate.calculate\_hash()

*if* block\_hash[:*self*.difficulty] == '0' \* *self*.difficulty:

*return* nonce

            nonce += 1

*def* validate\_chain(self):

*for* i *in* range(1, len(*self*.chain)):

            current\_block = *self*.chain[i]

            previous\_block = *self*.chain[i - 1]

*# Recalculate the current block's hash and compare*

*if* current\_block.block\_hash != current\_block.calculate\_hash():

                print(*f*"Invalid hash at block {current\_block.block\_id}")

*return* False

*# Check if the previous hash matches*

*if* current\_block.previous\_hash != previous\_block.block\_hash:

                print(

*f*"Invalid previous hash at block {current\_block.block\_id}")

*return* False

*return* True

*def* save\_chain(self):

*with* open(*self*.filename, 'w') *as* file:

            json.dump([block.to\_dict()

*for* block *in* *self*.chain], file, indent=4)

        print(*f*"Blockchain saved to {*self*.filename}.")

*def* load\_chain(self):

*with* open(*self*.filename, 'r') *as* file:

            chain\_data = json.load(file)

*self*.chain = [Block.from\_dict(block\_data)

*for* block\_data *in* chain\_data]

        print(*f*"Blockchain loaded from {*self*.filename}.")

*def* display\_chain(self):

*for* block *in* *self*.chain:

            print(*f*"Block ID: {block.block\_id}")

            print(*f*"Nonce: {block.nonce}")

            print(*f*"Data: {block.data}")

            print(*f*"Previous Hash: {block.previous\_hash}")

            print(*f*"Block Hash: {block.block\_hash}\n")

*def* menu():

    blockchain = Blockchain(difficulty=4)

*while* True:

        print("\nBlockchain Menu")

        print("1. Add a block")

        print("2. Display blockchain")

        print("3. Validate blockchain")

        print("4. Exit")

        choice = input("Enter your choice (1-4): ")

*if* choice == '1':

            name = input("Enter student's name: ")

            rollnumber = input("Enter student's roll number: ")

            branch = input("Enter student's branch: ")

            student = Student(name, rollnumber, branch)

            blockchain.add\_block(student)

            print("Block added successfully!\n")

*elif* choice == '2':

            blockchain.display\_chain()

*elif* choice == '3':

*if* blockchain.validate\_chain():

                print("Blockchain is valid.")

*else*:

                print("Blockchain is invalid.")

*elif* choice == '4':

            print("Exiting...")

*break*

*else*:

            print("Invalid choice. Please try again.")

*if* \_\_name\_\_ == "\_\_main\_\_":

    menu()

**Output:**

**View chain**

**A computer screen shot of a number and text

Description automatically generated**

**Add new block**

**A screen shot of a computer

Description automatically generated**

**A screenshot of a computer program

Description automatically generated**

**Validating via proof of work**

**A screenshot of a computer screen

Description automatically generated**