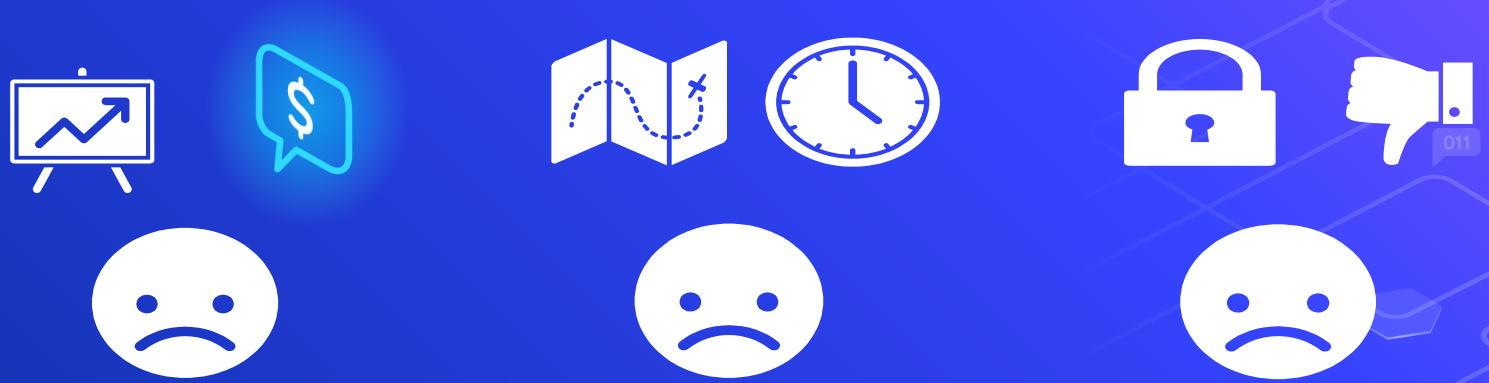
Implementing Blockchain as an Alternative to Academic Transcript

Group 2 - NULL
Members: Jonathan C., Andrew F., Angela T., and Vananh V.

Overview

- Recap: The Problems with Paper Transcripts
- Recap: Our Solution
- Peer-to-Peer (P2P) Network
- Our Blockchain Data Structure
- The GUI
- IT’S DEMO TIME!!!
- What’s next?
- Q&A

Recap:
The Problems with Paper Transcripts



Recap: Our solution

Peer-to-Peer (P2P) Network

By Jonathan & Andrew

Peer-to-Peer (P2P) Network Tools

Programming Language: 

Libraries:

- asyncio
- threading
- json
- hashlib

Connecting and Handling

```
async def handler(reader, writer):
    data = await reader.readuntil(EOF)
    message = unpacker(data)
    message_type = message[0]
    payload = message[1]

    if message_type == 'getblocks':
        task = asyncio.create_task(send_blocks(reader, writer, payload))
        await task
    else:
        print("Unknown message type received.")

    writer.close()


async def connect(port):
    # Connect to selected peer
    reader, writer = await asyncio.open_connection(
            '127.0.0.1', port)

    task = asyncio.create_task(get_blocks(reader, writer))
    await task

    writer.close()
```

Asynchronous:

- Non-blocking
- Event loop
- Coroutines
- Await

Getting Blocks

```
async def get_blocks(reader, writer):
    headers = []
    for block in reversed(BLOCKCHAIN):
        headers.append(block['header']['prev_hash'])

    payload = json.dumps(headers)
    message = package('getblocks', payload)
    writer.write(message)
    await writer.drain()

    while True:
        message = await reader.readuntil(EOF)
        message = unpacker(message)
        message_type = message[0]
        payload = message[1]
        if message_type == 'block':
            block = json.loads(payload)
            current_header_hash = hasher(block['header'])
            value = int(current_header_hash, 16)
            previous_header_hash = hasher(BLOCKCHAIN[-1]['header'])
            if block['header']['prev_hash'] == previous_header_hash and value < TARGET:
                BLOCKCHAIN.append(block)
                response = package('next', 'next block')
                writer.write(response)
                await writer.drain()
            else:
                print('Invalid block!')
                response = package('stop', 'no more')
                writer.write(response)
                await writer.drain()
                break
        elif message_type == 'finished':
            break
        else:
            print('Invalid message type!')
            break
```

Initial Block Download

- Block headers
- Response
- Validation
- Update the blockchain

Sending Blocks

```
async def send_blocks(reader, writer, payload):
    start = 0
    headers = json.loads(payload)
    try:
        for header in headers:
            for i, block in reversed(list(enumerate(BLOCKCHAIN))):
                if header == block['header']['prev_hash']:
                    start = i+1
                    raise BreakLoop
    except BreakLoop:
        pass

    if start < len(BLOCKCHAIN):
        for block in BLOCKCHAIN[start:]:
            payload = json.dumps(block)
            response = package('block', payload)

            writer.write(response)
            await writer.drain()

            message = await reader.readuntil(EOF)
            message = unpacker(message)
            message_type = message[0]

            if message_type != 'next':
                break

    response = package('finished', 'all done')
    writer.write(response)
    await writer.drain()
```

Block Transfer

- Compare headers
- Find start point
- Send block
- Await response

Data Structure

By Vananh Vo

Data Structure - Tools & Prog. Language

Programming Language: 

IDE: Pycharm 

Libraries: **hashlib**

Data Structure - Block

The design of our block:

- index
- previous_hash
- timestamp
- student_id
- course_id
- crade_earned
- unit_earned
- proof

```
from time import time

from utility.printable import Printable

class Block(Printable):
    """A single block of our blockchain.

    Attributes:
        :index: The index of this block.
        :previous_hash: The hash of the previous block in the blockchain.
        :timestamp: The timestamp of the block (automatically generated by default).
        :student_id: Student identification number.
        :course_id: Course identification number.
        :grade_earned: Grade earned by the student
        :unit_earned: Total unit(s) earned from the class
        :proof: The proof of work number that yielded this block.
    """
    def __init__(self, index, previous_hash, student_id, course_id, grade_earned, grade_earned, unit_earned, proof, time=time()):
        self.index = index
        self.previous_hash = previous_hash
        self.timestamp = time
        self.student_id = stuid
        self.course_id = course_id
        self.grade_earned = grade
        self.unit_earned = unit
        self.proof = proof
```

Data Structure - Using the hashlib library

The fundamental security structure of blockchain that is each block contain a hash of the previous block except for the first block.

```python
import hashlib as hl
import json

def hash_string_256(string):
    """Create a SHA256 hash for a given input string.

    Arguments:
        :string: The string which should be hashed.
    """
    return hl.sha256(string).hexdigest()


def hash_block(block):
    """Hashes a block and returns a string representation of it.

    Arguments:
        :block: The block that should be hashed.
    """
    return hash_string_256(json.dumps(block, sort_keys=True).encode())
```

The GUI

By Angela

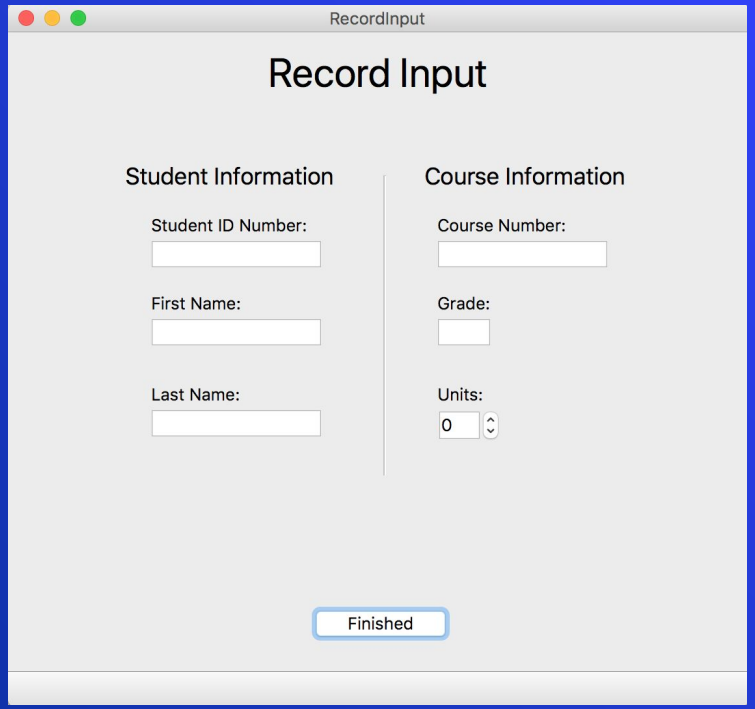
GUI - Tools

- Programming Language:

- PyQt5

- PyQt Designer

GUI - Record Input Window



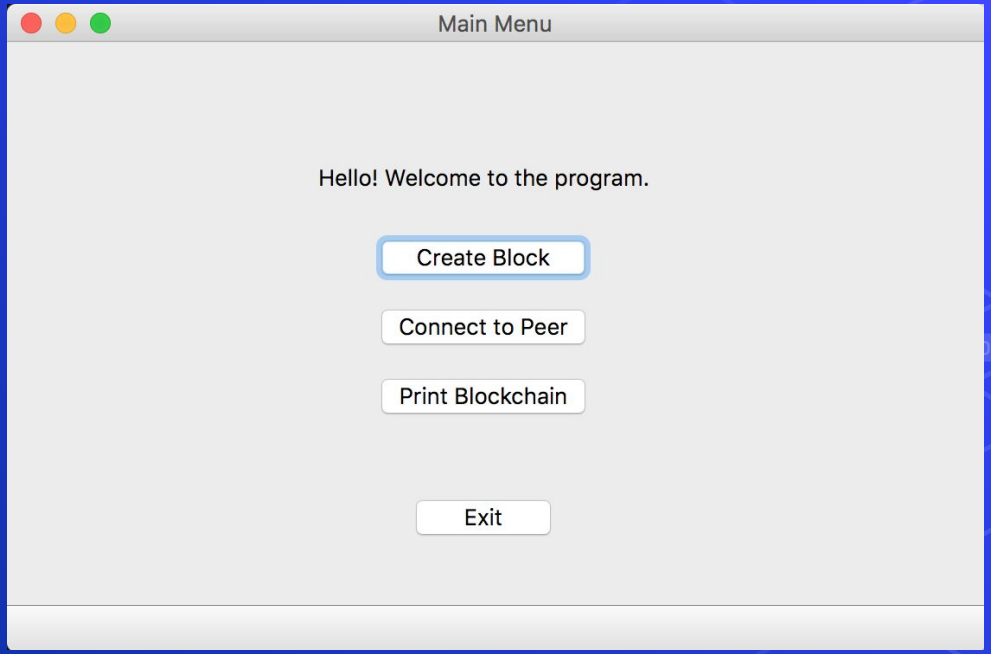

```python
from PyQt5 import QtCore, QtGui, QtWidgets


class Ui_RecordInput(object):
    def setupUi(self, RecordInput):…

    def retranslateUi(self, RecordInput):…

    def exitProgram(self):
        sys.exit()


if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    RecordInput = QtWidgets.QMainWindow()
    ui = Ui_RecordInput()
    ui.setupUi(RecordInput)
    RecordInput.show()
    sys.exit(app.exec_())
```




```python
from PyQt5 import QtCore, QtGui, QtWidgets
#import async_protocol


class Ui_MainWindow(object):
    def setupUi(self, MainWindow):…

    def retranslateUi(self, MainWindow):…

    def exitProgram(self):
        sys.exit()

    def add_block(self):
        x = input("Name: ")
        print(x)


    def testFuct(self):
        _translate = QtCore.QCoreApplication.translate
        x = "HAHA I CHANGED IT!"
        self.welcomeMessage.setText(_translate("MainWindow", x))


if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    MainWindow = QtWidgets.QMainWindow()
    ui = Ui_MainWindow()
    ui.setupUi(MainWindow)
    MainWindow.show()
    sys.exit(app.exec_())
```

It’s Demo Time!
Let’s smash ’em up!

What’s next?

Questions?

Any question? Come on, we know you are curious! 😉