FINAL ASSIGNMENT Part 1

|  |
| --- |
| BLOCKCHAIN TECHNOLOGY (INFBCT01) |

Educational Period 1&2 [2025-26]

|  |  |
| --- | --- |
| **GOODCHAIN** | An Asynchronous Blockchain Wallet/Miner App |  |

The main objective of this assignment is to integrate all the previous tutorials and weekly practicums to build a simple custom tamper-proof ledger which would be stored on the local machine. In fact, we will create a proof of concept of a local blockchain-based wallet/miner simulator to allow users to perform transactions on their wallet, and mine new blocks, view the ledger, etc.

Please note that the scenario described here, including the components, terminologies, and functionalities, might be different from a real-world application of a blockchain or similar applications. This custom-made application is defined only for the learning purpose in this course.

## Intended Learning Outcome:

1. To understand fundamental concepts in blockchain (private key, public key, digital signature, hash, transaction, public ledger, block, tamper-proof blockchain, decentralization, mining, consensus, proof of work, nonce, mining reward, transaction fee, verification, validation and storage of transactions and blocks).
2. To develop cryptographic concepts in blockchain (private key, public key, digital signature, hash)
3. To develop basic components of a blockchain and cryptocurrency (transaction, public ledger, block, and tamper-proof blockchain) and their operations (verification, validation, load, and store).
4. To implement decentralization, mining, consensus, proof of work, nonce, mining reward, transaction fee.

|  |
| --- |
| Description of the System |

## 

## General Overview

**Goodchain** is an asynchronous multi-user wallet/miner simulator which is implemented on a blockchain data structure. This assignment consists of the design and implementation of a console-based application in Python 3 for the described system. It should use a local database to store the data of users (for example: username, password hash, private key, and public key). There is no KYC (Know Your Customer), so no other information of a user should be asked or stored.

In Goodchain, the blockchain grows based on the consensus of the users, so there is no central authority or administrator to take care of the system. In the first stage (i.e., Final Assignment Part 1) It does not run on a P2P network, meaning that the users do not simultaneously login and work. Instead, only one user can run the application or login to the system at a time and perform some activities. Once a user logs in to the system, they can act as a node to carry out multiple functions, including checking the status of the blockchain, checking the current transactions on the waiting transactions pool, mining a new block, transferring coins to other registered users, and contributing to the consensus process, if needed. We call the special consensus of this blockchain as “asynchronous consensus”, opposed to real-time consensus in P2P.

Once a new user is signed up in the system, an offline node is created on the system. Then, the user will have a private key and a public key associated with his/her newly created wallet. The public key will act as their account number on the blockchain, and the private key will be used for authorization of financial transactions on their wallet.

There are three main activities (plus some other functions) for a logged-in user (node):

1. **Transfer coins**

A user can transfer some coins to any other registered user on the system.

1. **Mine a new block**

Providing specific conditions, a user can mine and create a new block and add it to the blockchain. If the block is accepted on the blockchain by consensus of other users, then the miner will receive a mining reward.

1. **Explore the blockchain**

The blockchain is fully transparent to all participants, and every user (including registered or non-registered users) can always check the status of the blockchain, and view all data stored on the blockchain.

These activities are explained in more detail in the next pages.

In addition to these three main activities, a logged-in user can also check their balance and see the current pool of unprocessed transactions. They can also cancel their own transaction(s) on the pool if they are not processed yet.

A general overview of the system is depicted in Figure 1.

Diagram

Description automatically generated

**Figure 1.** General overview of the system. No more than one user can be simultaneously active (logged-in).

An example of the system menu for a non-logged-in user (public menu) and a logged-in user (node menu) are depicted in Figure 2.

|  |  |
| --- | --- |
|  |  |
| Public menu (not logged-in user) | Node menu (logged-in user) |

**Figure 2.** Public menu vs Node menu

## Requirements

The final product must fulfill at least the following conditions and requirements:

### General requirements:

* The system must be built based on blockchain data structure.
* There is no administrator or central authority to manage or control the system.
* There are two types of users: public users and registered users (nodes)
* A user can add a new block to the chain if the block is successfully mined.
* When a transaction is created by a user, the transaction will be placed in a transactions pool.
* A user can always check the validation of the whole chain.
* A user can always check the validation of a specific block and add a validation flag to a valid block.
* The system must always validate the blockchain for any tamper on every reading or writing or any other operation on the blockchain.
* A user can see the data of every block.
* A user can see the information of the whole blockchain, including the number of blocks and the total number of transactions.
* A user can cancel of modify their own pending transactions on the pool.

### User interface:

* The application must have a textual console-based user interface
* The user interface must have a menu system to facilitate the operation of the application.
* The system should always keep users informed about what is going on.
* User should be able to see the status of and some general information about the system, including the transactions, pool, blockchain, their profile, notifications, recent changes on the system, ongoing actions, etc.
* The UI should support undo and redo actions when entering data.
* User should have sufficient control and freedom on the system.
* Users should not have to wonder whether different words, situations, or actions mean the same thing.
* There must be reasonable communication with the user regarding the status of the system.
* Use should have enough contextual help related to the operations of the system.

### Public user

A public user is a user who is not registered or signed up in the application. This user has some limited access to the system.

* Any user can freely register (sign up) in the application.
* Any user can explore the public ledger and view the data of each block.

### Registration (Sign up):

Once a public user signed up in the application, a node is created for the user. We will call this user a “node user”, or “logged-in user”, or only a “node” for short.

* A user must provide a unique username and a password when registering in the system.
* A node user will receive 50 coins as a sign-up reward, after registration.
* A unique pair of private key and public key must be created for a node user, after registration.

### Node user (Logged in user)

This user can login to the system and perform some specific activities, such as transferring coin, mining a block, etc.

* A node user has a wallet.
* A node user can send some coins from their wallet to the other registered users in the system.
* A node user can receive some coins in their wallet from the other registered users in the system.
* A node user can try mining a new block.
* A node user can see their balance on the user page.
* A node user can see the history of their own transactions thorough a menu.
* A node user can view the current ongoing transactions on the pool

|  |
| --- |
| **Note:** We will use the words “logged-in user”, “node user” or “node”, interchangeably in this assignment. Once a public user signs up in the system, they make a new node. However, we use the word “logged-in user” whenever a function of the system must be manually or intentionally initiated by the user, such as mining a block or transferring some coins. On the other hand, there are some activities which do not need to be started by the user, instead will be automatically executed by a node (on behalf of the logged in user). For such cases, we use the term “node”. For example, once a user logs in to the system, the node must check if there are any pending blocks for verification. This is automatically done by a node, and the logged in user does not need to initiate it. |

### Ledger (Blockchain data structure):

* Each block has a unique ID (a sequential number starting from Zero for the genesis block).
* Every block must contain a minimum of 5 and a maximum of 10 transactions, including the reward transactions.

### Transactions:

* An extra value could be placed on a transaction as transaction fee.
* A sender must enter the transaction fee, while creating a transaction.
* The transaction fee is incentive to motivate miners to pick a transaction for their new block.

### Data files

As shown in the Figure 1, there are three data files required for the system to meet the required functionalities. The purpose of each file is briefly described below:

* **The Database**

A relational database is required to store data of the registered users, including username, hash of password, private key, and public key, and more information if needed (e.g., recovery phrase key). It is obvious that this database must be securely implemented. This database must be used only for the users’ information and no data from the blockchain or the pool should be stored in the relational database.

* **The Ledger**

This is the main file to store all data of transactions. It must be developed according to requirements of blockchain data structure, including all the required components (for example, hash of block, metadata, transaction data, nonce, etc.)

* **The Pool**

As this application can serve only one user at a time, some changes of the blockchain (e.g., requests for transactions) needs to be temporarily placed in a file, until the validation process or consensus process is completed. The pool file is mainly used for this synchronization purpose.

This file contains a list of transactions, and its format must be according to the structure of the transactions (the same format with the ledger). Do not use a relational database, JSON, csv, XML, or any other similar standard data transmission protocols or format. (In the practicum 6, we have already discussed and learned how to save and load transactions on a file).

In the final assignment part 1, we have only **one instance of each file**, shared among all users (only one ledger, one user database, and only one pool). As at any specific time, only one user can access the file, it is not needed (and not allowed) to have a separate copy of files for each node.

### Mining:

* A new block could be mined, if there are a minimum of 5 valid transactions on the pool.
* A new block could be mined after the last block, only if every previous block in the chain has at least 3 valid flags.
* A block must be mined between 10 to 20 seconds. A proof of work is implemented for this purpose, which creates a specific hash with a specific number of leading zero. (This timing is for testing while assessing and grading your code. Do not use any sleep or delay function to adjust the time if it is too fast).
* A minimum of 3 minutes interval must be between every two consequent blocks.
* A miner will receive 50 coins, as a mining reward for a successful block added to the chain.

### Security:

* SHA256 must be used for any hashing in the system.
* A password must be saved in the form of a hash in the system.
* A user must be able to see their private key and public key when logged in.
* A username (or hashed unique public key) must be used as the public account number of a user for any transaction.

There are other requirements explained in more detail, later, in the description of specific functions of the system, such as setup, mining, coin transfer, notifications, etc.

## More Details about the Requirements

### Setup of the system at startup

When the system starts, it should check the data files shown in Figure 1 and initialize the system. These data files include a relational database to store data of registered users, the ledger (the blockchain), and the pool (a storage to temporarily keep the transaction until they get added to the ledger).

There should be a comprehensive check (and configuration, if needed) of the data files. in the next section, you can find more details about the required data files.

It must also perform a check for any tamper on any of these files. Any change from the outside of the system should be recognized.

### Transfer some coins (transaction)

To transfer a coin, a user must create a new transaction. The transactions should be validated twice in the system:

1. **When the transaction is created by the user**

The system should not allow a user to place an invalid transaction in the transaction creation process. When creating a transaction, the system needs to correctly respond to the user’s activity. For example, if the transaction is valid, and there is sufficient fund in the user’s account, the transaction order must be placed in the pool, waiting for the next mining attempt. On the other hand, if there is no sufficient fund or the transaction is invalid (for other reasons), it must prevent creating such an invalid transaction.

1. **When a node is trying to mine a block**

If an attacker could successfully tamper a transaction on the pool, a miner node must be able to recognize such an invalid transaction. In this case, the transaction should be flagged as invalid by the miner. When the creator of the transaction logs in to the system, the flagged transaction must be automatically canceled by the node.

### Mining

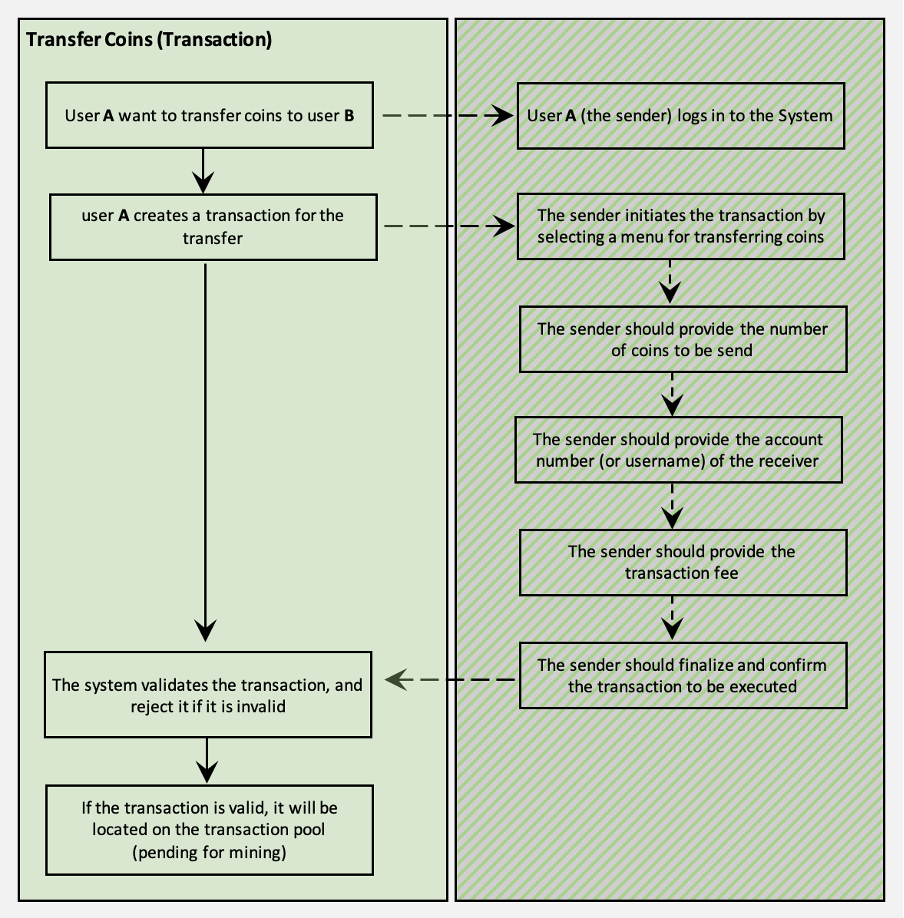
Once a user logs in to the system, they can decide to mine a new block and receive reward for the proof of work. For this, the node should check the pool for valid transactions, and find a minimum of 5 and a maximum of 10 valid transactions to be added to the new block. A motivation for a node to include a transaction in their new block for mining is the transaction fee. A node is free to have a strategy to choose specific transactions and how many transactions to be added to the new block. However, the strategy must ensure that the selection process is not biased to a specific node. It means that for example a node cannot wait only for its own transactions or always ignore the transactions with zero transaction fee or lower fee. It can choose the highest fee first, but if there are some other transactions in the pool, they need to be finally included in a block.

During this process, a miner will check the transactions and validate them. Any invalid transaction must be flagged as invalid, in the pool. These invalid transactions must be automatically canceled and removed from the pool by the creator of the transactions upon their next login.

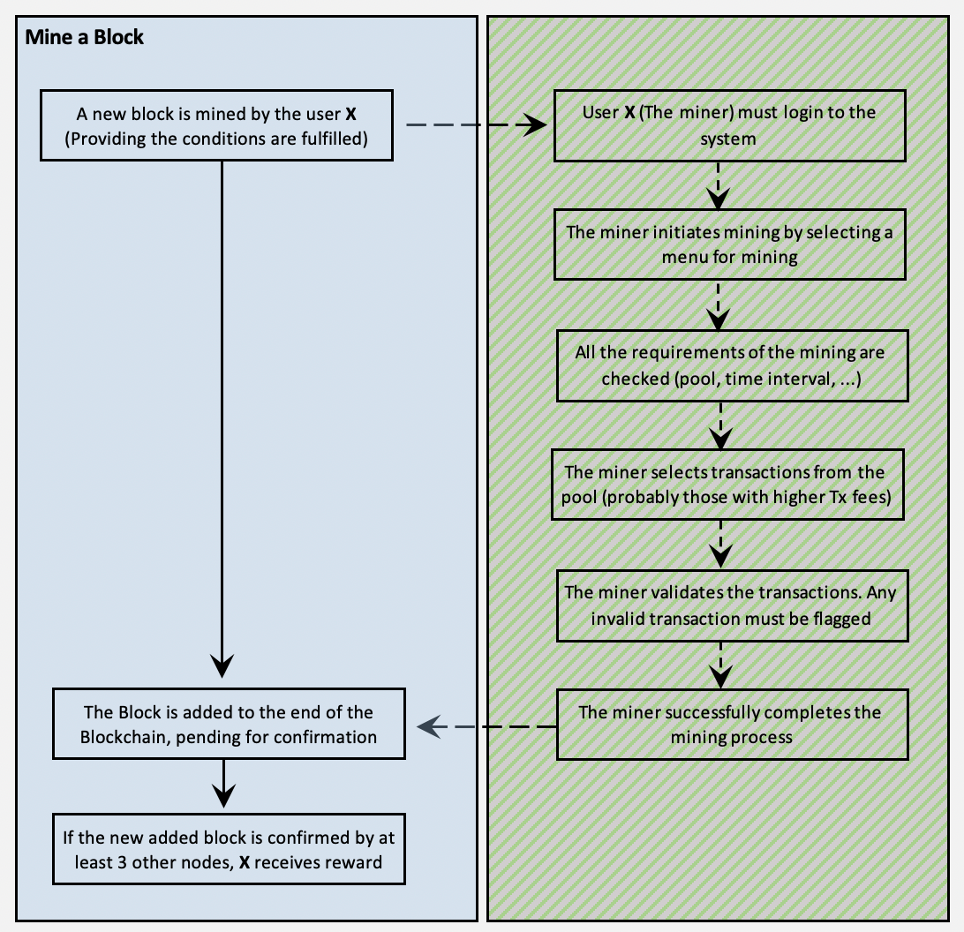
Once a block is created by a node, the next three logged in users (nodes) must check the validity of the created block. These nodes will fully check the block to ensure a valid block is created by the miner. If the block is valid, they flag it as valid, otherwise they flag it as invalid.

* If the new block is flagged as valid by these three nodes (three different logged in users), then the third validator node is responsible to create a new transaction to reward the miner of the block. This reward transaction could be included in the next mining process. If a block successfully got validated by three other nodes, it does not need to be validated by any other nodes later. (Note that the block cannot be validated by the creator of the block.)
* If the new block is flagged as invalid (rejected) by at least three other nodes, before getting three valid flags, then the third rejector node is also responsible to return all the transactions of the rejected block back to the pool. In this case, if the block is rejected because of some invalid transactions, those invalid transactions must be also flagged as invalid on the pool to be nullified by the creator of the transaction upon login. Other valid transactions in the rejected block must be returned to the pool, waiting for the next mining process to be included in a new block again.

A general overview of the coin transfer and mining processes are briefly shown in Figures 2 and 3.



**Figure 2.** Overview of the coin transfer process.



**Figure 3.** Overview of the mining process.

### Notifications when a user logs in

When a user logs in to the system, appropriate notifications should be displayed to the user. These notifications could include any relevant information and changes on the status of the blockchain and pool, plus any other relevant pending activities. Some of the important required notifications are listed below:

* General information of the blockchain (the size of blockchain, number of transactions, etc.)
* Users’ mined block status (if a user already mined a block and the block was on pending for verification by other nodes
* Any block which was on pending and is confirmed or rejected by this user after login
* Reward notification if there was any reward pending for confirmation from other nodes
* New added block(s) since the last login (already confirmed by other nodes or waiting for a confirmation)
* Rejected transactions of the user
* Successful transactions of the user

You may need to add more notifications.

### Automatic functions of nodes upon login

Once a user logs in to the system (which means a node is active), the node must perform some specific actions, if required. These activities are explained in detail in different sections of this system.

For example, if there is a new block created by another node, but it is not yet validated by enough nodes and not finalized, then the logged in node must automatically start a validation process of the new block. Another example is if earlier a transaction was ordered by this node, and the transaction was rejected for some reason and flagged as invalid, the transaction must be canceled by the node. There might be some other functions which need to be automatically managed by a node.

|  |
| --- |
| Submission |

## 

## Test the system before submission

Before you submit the deliverable package, test the product with the following test cases, in the given order. This gives you an idea of how the system will be checked during the grading process.

Please note that to assess the system functionalities, we need you to precisely carry out the following tests, with the given data and the given order. With these tests, we can track the system performance.

#### Register some users

To test the system, you need to initially register some users. For this purpose, create the following 3 users. Upon registration, a user will automatically receive 50 coins as a reward. This reward should be created as a form of a transaction waiting for the next mining process.

|  |  |  |
| --- | --- | --- |
| **Username** | **Password** | **Initial Balance (reward)** |
| mike111 | mike111 | 50 |
| rose222 | rose222 | 50 |
| alex333 | alex333 | 50 |

#### Try mining a new block

Try to mine a block, while there is not enough transactions in the pool. In this case, it is not feasible to mine a block, as there is not a minimum required number of transactions waiting in the pool. The system needs to properly display a message to the user and notify them.

#### Register more users

Add more users to the system.

|  |  |  |
| --- | --- | --- |
| **Username** | **Password** | **Initial Balance (reward)** |
| mara444 | mara444 | 50 |
| tina555 | tina555 | 50 |

#### 

#### Create some transactions

Try to execute the transactions listed below, in the given order. Any invalid transaction should be recognized and nullified by the system at the time of creation.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Transactions List 1** | | | | |
| **Tx.**  **No.** | **Sender** | **Receiver** | **Tx.**  **Value** | **Tx.**  **Fee** |
| **Username** | **Username** |
| 1 | mike111 | rose222 | **3.5** | **0.1** |
| 2 | alex333 | rose222 | **5.0** | **0.0** |
| 3 | mike111 | queen (invalid) | **2.0** | **1.0** |
| 4 | mike111 | mike111 | **5.0** | **0.0** |
| 5 | mara444 | alex333 | **-1.0** | **0.0** |
| 6 | mara444 | rose222 | **1.0** | **-1.0** |
| 7 | mara444 | mike111 | **25.0** | **0.5** |
| 8 | mara444 | mike111 | **0** | **0** |
| 9 | tina555 | mara444 | **0** | **1.0** |
| 10 | alex333 | mara444 | **120** | **100** |

#### Try mining a new block

Now, there are transactions on the pending list in the pool, but there are some invalid transactions. In this test, the system needs to properly validate all the pending transactions, flag the invalid transactions, and check if there are sufficient valid transactions in the pool.

In this test, the user can mine a block and add it to the blockchain.

#### Login to the system by other three users and confirm the mined block

Once a block is mined and added to the blockchain, the block will be on pending status for verification by consensus of other nodes. It needs at least three valid flags to become a valid block on the chain. Then, the miner could receive their reward, and a new block could be added after the block in the next mining process.

#### Create more transactions

Try to execute the transactions listed below, in the given order.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Transactions List 2** | | | | |
| **Tx.**  **No.** | **Sender** | **Receiver** | **Tx.**  **Value** | **Tx.**  **Fee** |
| **Username** | **Username** |
| 11 | tina555 | alex333 | **5.0** | **1.0** |
| 12 | tina555 | mike111 | **5.0** | **200** |
| 13 | alex333 | rose222 | **1.5** | **0.4** |
| 13 | rose222 | tina555 | **6.0** | **4.0** |
| 15 | tina555 | alex333 | **4.0** | **1.2** |

#### Try mining a new block

#### Login to the system by other three users and confirm the mined block

#### Create more transactions

Try to execute the transactions listed below, in the given order.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Transactions List 3** | | | | |
| **Tx.**  **No.** | **Sender** | **Receiver** | **Tx.**  **Value** | **Tx.**  **Fee** |
| **Username** | **Username** |
| 16 | mara444 | mike111 | **2.6** | **0.1** |
| 17 | rose222 | tina555 | **4.0** | **1.0** |
| 18 | rose222 | mike111 | **1.0** | **0.1** |
| 19 | tina555 | alex333 | **4.5** | **0.3** |
| 20 | rose222 | mara444 | **0.1** | **0.0** |
| 21 | alex333 | tina555 | **17.0** | **2.0** |
| 22 | mike111 | tina555 | **2.0** | **0.2** |
| 23 | tina555 | mara444 | **4.0** | **0.0** |
| 24 | mike111 | mara444 | **2.1** | **0.1** |
| 25 | mike111 | alex333 | **4.4** | **0.4** |
| 26 | mara444 | mike111 | **1.2** | **0.0** |
| 27 | mara444 | alex333 | **50.0** | **5.0** |
| 28 | tina555 | mike111 | **2.0** | **0.0** |
| 29 | alex333 | mara444 | **0.0** | **0.5** |
| 30 | alex333 | mike111 | **1.0** | **0.4** |

## What to deliver?

The delivery to be handed in must consist of one zip-file, named as below:

**Studentnumber1\_ Studentnumber2\_firstname1\_lastname1\_ firstname2\_lastname2.zip**

For example:

**1019972\_0990112**\_**Geert\_Jan \_Lexmond\_Marco\_de\_Boer**.**zip**

**The zip-file must contain:**

A main directory called **goodchain** containing 2 sub-directories as below:

1. A sub-directory called **src**, containing all the **code files**, including one main file **goodchain.py**. Starting the system should be done by running **goodchain.py**.
2. A sub-directory called **data**, containing all the **data files**.

Please note that the code must read/write only from/on the goodchain directory and its subdirectory, if needed. Do not attempt to read or write on any other locations on the disk.

## How to deliver?

The submission will be through the assignment channel of the course team in MS Teams.

## Deadline

You have a maximum of two chances for each part of the final assignment to get “sufficient” grade.

There are 4 chances in total for the submission of the final assignments, as given in the table below. Red cells show “insufficient”, and green cells show “sufficient”.

|  |  |  |  |
| --- | --- | --- | --- |
| 2025-2026 | | | |
| **OP1** | **OP2** | **OP3** | **OP4** |
| FA1 (1st) | FA2 (1st) |  |  |
| FA1 (1st) | FA1 (2nd) | FA2 (1st) |  |
| FA1 (1st) | FA2 (1st) | FA2 (2nd) |  |
| FA1 (1st) | FA1 (2nd) | FA2 (1st) | FA2 (2nd) |

If you could not get sufficient through any of the given paths, you need to retake the course, and resubmit all different components, again. The submission deadline will be announced in the course team on MS Teams.

|  |
| --- |
| ⚠ IMPORTANT NOTES  1. **Do not** include any **bulky** Python system files in the delivery. 2. In this assignment, in addition to the **standard library modules** of Python, you canuse **cryptography, pickle, sqlite3, random, re, sys, os,** andany third-party package for functions related to the user interface. If you need to use any other extra package, you may ask your teacher to check if it is allowed. 3. The code must run **error-free** (on a standard Windows or MAC PC). |

|  |
| --- |
| Grading |

## 

## Marking Scheme

There are different criteria to be checked during the assessment. These criteria are categorized for more clarity and readability.

|  |  |  |
| --- | --- | --- |
| **C5.1: User Interface (Node)** | | **Some Examples of Good Practices and Satisfactory Implementation:** |
| C5.1.01 | Visibility of system status | The system should always keep users informed about what is going on. For example, the progress during a file transfer. Provide immediate feedback, etc.  User can see the status of and some general information about the system, including the transactions, pool, blockchain, their profile, notifications, recent changes on the system, ongoing actions, etc. |
| C5.1.02 | User friendly design | User can easily work with the user interface.  There is a simple textual user menu, which allows a user to efficiently use the system for various activities. |
| C5.1.03 | User control and freedom | Supports undo and redo actions. Allow user to override the system.  User has sufficient control and freedom on the system. |
| C5.1.04 | Consistency and standards | Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow behavior user except and platform conventions. |
| C5.1.05 | Error prevention and management | Prevents a problem from occurring in the first place.  There is a good level of input validation.  Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution. |
| C5.1.06 | Recognition rather than recall | Making objects, actions, and options visible.  Make navigation visible and make it easy to go back, go to home page and quit. |

|  |  |  |
| --- | --- | --- |
| **C5.2: Functionality (Node)** | | **Some Examples of Good Practices and Satisfactory Implementation:** |
| C5.2.01 | User management | The system allows a user to register, edit and update their profile, login to the system, etc.  The user account is secured with a hashed password and other security policies.  The passwords are not stored in the database.  The data of the users are securely stored in a database. |
| C5.2.02 | Key management | Private keys and public keys of users are correctly and securely handled and stored.  There is an offline secure recovery phrase system for private key. |
| C5.2.03 | Balance check | A user can easily see the balance and trance it back over the time. |
| C5.2.04 | Notifications | There is a notification system which provide required information for the user whenever something concerned to the node or user is happened. For example, if a new block is already added to the blockchain, or if a node can mine a new block, or verify a block which is already mined by another node, or a user’s transaction is rejected or added to a block, or if a user received some coins. |
| C5.2.05 | Decentralization | The system design supports decentralization.  All users have the same accessibility and control over the system.  There is no central authority. There is no central database.  All nodes have the latest copy of the user’s database, blockchain and pool. |
| C5.2.06 | Transfer coin | User can send/receive coins to/from other users (with option to use their public key or username).  A valid transaction can be created by any user.  A node checks the validity of a transaction.  A valid transaction can be added to the pool. |
| C5.2.07 | Pool explorer | Node user can explore the pool and view the transactions placed on the pool. |
| C5.2.08 | Block verification | Node user can validate a block, waiting for verification (consensus).  Node user can validate any block on the ledger, at any time, for any tamper. |
| C5.2.09 | Mining | Node user can try mining a new block.  Providing the conditions for mining a new block, a node user can mine a new block. |
| C5.2.10 | Chain explorer | Node user can explore the ledger and view the transactions placed on the pool.  There are different options for exploring the ledger: 1) view a specific block by its number, 2) view all blocks from the genuine block to the last block, with possibility to see them page by page, 3) view only the last block. |

|  |  |  |
| --- | --- | --- |
| **C5.3.: Public and Private Keys** | | **Some Examples of Good Practices and Satisfactory Implementation:** |
| C5.3.01 | Creation of keys | Private and public keys are correctly generated.  An appropriate module or library (with correct configuration) is used for the generation of keys. |
| C5.3.02 | Load and store | Keys are correctly managed (load and store) on the storage device. |
| C5.3.03 | Security of keys | Keys are securely managed (load and store) on the storage device. |

|  |  |  |
| --- | --- | --- |
| **C5.4: Digital Signature** | | **Some Examples of Good Practices and Satisfactory Implementation:** |
| C5.4.1 | Sign a transaction | A transaction can be signed. |
| C5.4.2 | Signature verification | A transaction can be verified. |

|  |  |  |
| --- | --- | --- |
| **C5.5: Transaction** | | **Some Examples of Good Practices and Satisfactory Implementation:** |
| C5.5.1 | Data structure of transaction | The data structure of transaction is well defined. |
| C5.5.2 | Creation of transaction | A transaction can be created and added to a block. |
| C5.5.3 | Validation of transaction | A transaction can be validated. |

|  |  |  |
| --- | --- | --- |
| **C5.6: Blockchain (the Ledger)** | | **Some Examples of Good Practices and Satisfactory Implementation:** |
| C5.6.1 | Data structure of block | The data structure of transaction block is well defined. |
| C5.6.2 | Data structure of blockchain | The data structure of blockchain is well defined. |
| C5.6.3 | Tamper-proof feature | A block is secure and tamper-proof. |
| C5.6.4 | Validation of blockchain | A block can be validated for any change. |

|  |  |  |
| --- | --- | --- |
| **C5.7: Blockchain Storage** | | **Some Examples of Good Practices and Satisfactory Implementation:** |
| C5.7.1 | Load and store | Blockchain file can be loaded and stored appropriately. |
| C5.7.2 | Data file structure | Blockchain data file structure is appropriate. |

|  |  |  |
| --- | --- | --- |
| **C5.8: Mining** | | **Some Examples of Good Practices and Satisfactory Implementation:** |
| C5.8.1 | Validation of transaction block | Transaction block can be validated. |
| C5.8.2 | Proof of work | The proof of work (using nonce) is correctly implemented. |
| C5.8.3 | Mining process time | The mining time is appropriate. |
| C5.8.4 | Time interval between two mining | The time interval between to consecutive mining is appropriate. |

|  |  |  |
| --- | --- | --- |
| **C5.9: Pool** | | **Some Examples of Good Practices and Satisfactory Implementation:** |
| C5.9.1 | Data structure of pool | The data structure of pool is well defined. |
| C5.9.2 | Pool management | The transactions could be managed on the pool. The file format is appropriate. Load, store, and update are properly implemented. |

|  |  |  |
| --- | --- | --- |
| **C5.10: Presentation** | | **Some Examples of Good Practices and Satisfactory Implementation:** |
| C5.10.1 | Explore and demonstrate the work | Student is able to run the work and show the functionality of the system.  Students is able to show the code and explain various parts of the code and their functions. |
| C5.10.2 | Answer to questions | Students is able to answer questions about the system. |
| C5.10.3 | Originality of the work | Student is able to show the originality of the code, and his/her contribution by providing reasons and justifications for various sections of the code and their purpose and functionalities |

|  |  |  |
| --- | --- | --- |
| **C5.11: General Functionality** | | **Some Examples of Good Practices and Satisfactory Implementation:** |
| C5.11.1 | Major Error? | The system must run smoothly.  The submitted code must be error-free, means that there must not contains any serious error to prevent the normal execution of the code or lead to a code crash. |
| C5.11.2 | Major Bug? | The system must not contain any serious bug which influence the requirements of the system. (For example, if the system wrongly calculates the balance, it contains a bug). In such cases. Even the system works without error, but a major bug is not accepted. |

## Presentation

To fairly assess your assignment, we ask you to present your work. With this we ensure that your code will run smoothly, and we can see all the requirements and functionalities of your code without confusion and avoid any mis-grading of your assignment.

The presentation will be placed in the class time. You will receive your grade and feedback, after your presentation.

Each student/group has 50 minutes (approximately 20 minutes for the presentation, and 30 minutes for the teachers for questions, grading, discussion, and feedback).

## Sample submitted works

There are two sample works from previous cohorts. These are just samples to give you a general idea about the assignment requirements and does not mean that these works are strong or weak.

* [Video 1](https://hrnl-my.sharepoint.com/:v:/g/personal/bashb_hr_nl/EUWOW5PylEFHovmi2ZLzdckBXIWVAttmp6wlYHLtsgT10A?e=TZEvor) – [Mining Demo](https://hrnl-my.sharepoint.com/:v:/g/personal/bashb_hr_nl/EcFTGGQFQbZOqPWtZLhd83UBWWeM1eNlkvf3pG3i0IG8tw?e=FegIvb)
* [Video 2](https://hrnl-my.sharepoint.com/:v:/g/personal/bashb_hr_nl/EZo5iw75N1tOmiJY8whNTf8BkuK5AH_n_Fpt9pmgK9kp4A?e=yjPEPR)