Année académique 2020-2021

Promotion : Bachelor 2 Informatique Développement

CAMPUS CONTEST

B2 Développement

Objectif du Campus Contest

Ce brief projet s'adresse à un développeur d'application qui devra mettre en œuvre toutes les compétences acquises durant son apprentissage pour concevoir une ébauche de blockchain.

Calendrier Campus Contest

Une période est dédiée à ce projet dans votre calendrier :

Les 8 et 9 Juillet 2021.

Le travail devra être rendu au plus tard le 9 Juillet, à 17h. Il est à envoyer sur l'adresse mail : **k.niel.pro@gmail.com** . Le mail de rendu devra être professionnel et ne devra contenir QUE le lien de votre repository github.

NB : Etant donné les quelques soucis que nous avons déjà rencontré par le passé, en cas de non réception du travail de votre part, j'essaierais de vous avertir le plus rapidement possible. N'hésitez pas à vérifier vos mails dans les jours qui suivent afin d'être sûr de ne pas avoir de requête de ma part à ce sujet.

Modalités

Projet individuel, aucun travail de groupe.
Responsabilité individuelle pour mettre en œuvre les compétences algorithmiques et de programmation orienté objet.

Evaluations

Vous trouverez le barème d'évaluation à la fin de ce document.

1. Contexte du projet

La technologie Blockchain est de plus en plus utilisée dans de nombreuses thématiques. Il est donc primordial de s'y intéresser en tant que développeur et de comprendre à minima les mécaniques de base. Vous devrez donc, à l'aide des indications que vous trouverez ci-dessous, réaliser une ébauche de blockchain fonctionnelle.

1.1 Langage de programmation

La technologie blockchain est connue pour être gourmande en ressources, vous devrez donc utiliser le langage Python afin de réaliser ce projet. Celui-ci étant notamment réputé pour sa rapidité de réalisation de calculs, il est donc tout indiqué.

Vous devrez par ailleurs exploiter l'ensemble de vos connaissances en Programmation Orientée Objet afin de réaliser ce projet.

1.2 Livrables attendus

Vous devrez livrer un repository github ou gitlab contenant l'ensemble de votre travail. Celui-ci devra par ailleur contenir :

- Le code source de votre blockchain
- Le contenu de la blockchain
- Un readme expliquant son fonctionnement et comment l'utiliser

1.3 Informations supplémentaires

Voici quelques indications supplémentaires que vous devrez garder en tête tout au long du projet :

- Vous devrez respecter la trame ci-dessous, incluant le nom des attributs et le nom des méthodes donnés
- Vous devrez respecter au maximum la **PEP8**, que vous pourrez vérifier grâce à l'outil **flake8**.
- Les méthodes et classes imposées ci-dessous peuvent, si vous le jugez nécessaire, inclure différents paramètres. C'est à vous de les définir en fonction de vos besoins !
- De manière générale, et malgré l'utilisation de python, créez un fichier par classe. Ne réalisez pas plusieurs classes dans un seul fichier
- Si certains éléments vous semblent manquants, ou certaines indications floues, c'est que vous êtes libre d'implémenter une solution comme bon vous semble. En cas de doute, dirigez vous systématiquement vers le bon sens, et détaillez votre logique et vos explications dans un commentaire
- **Il est fortement recommandé de lire l'ensemble de l'énoncé AVANT de commencer à coder : si vous n'avez pas une bonne vue d'ensemble, vous risquez de rencontrer des difficultés.**
- Prenez bien en compte l'ensemble du barème d'évaluation, gardez encore une fois en tête que c'est le critère principal de réussite d'un projet
- A chaque fois que vous arrivez à franchir une étape, pensez à réaliser un commit, et à le nommer correctement !

2 Mise en place du projet

2.1 Mise en place de l'environnement de travail

2.1.1 - Initialisation d'un repository github en mode "public" : Attention à bien vérifier que votre repository soit visible même pour une personne non connectée !

2.1.2 - Création d'un dossier sur l'ordinateur, initialisation de GIT et mise en lien avec le repository distant. Vous pouvez utiliser la méthode que vous souhaitez pour réaliser cette étape.

2.2 Initialisation du projet en python

2.2.1 - Création d'un fichier principal : "main.py". Ce fichier devra être le point d'entrée de votre programme, et se trouver à la racine de votre projet.

2.2.2 - Création d'un dossier "classes", dans lequel seront stockés les fichiers contenant des classes. Vous devez transformer ce dossier en "package" python.

2.2.3 - Création d'un package python "settings" qui contiendra les fichiers de configuration.

2.2.4 - Création d'un dossier "content", qui contiendra le contenu de votre blockchain.

2.3 Mise en place de la classe principale

La blockchain est, comme son nom l'indique, une chaîne (ou "suite") de blocs, c'est-à-dire que plusieurs "blocs" existent, sont reliés les uns aux autres, et constituent tous ensemble une chaîne. Afin de commencer la réalisation de votre blockchain, vous devrez donc créer une première classe, que vous pourrez appeler "Chain", qui représentera l'ensemble de la chaîne.

Vous pouvez ensuite y intégrer un attribut "blocs", qui contiendra une liste (ou un dictionnaire) de blocs.

2.4 Mise en place de la classe de gestion des utilisateurs

Un des principes de la blockchain consiste en un partage complet des informations de la chaîne envers tous ses utilisateurs. Ainsi chaque utilisateur devra disposer d'un identifiant unique au sein de la chaîne afin d'être clairement identifié. Ce numéro d'identifiant unique est généralement appelé "wallet" (portefeuille) et un utilisateur n'est donc pas identifié en tant qu'utilisateur, mais par le portefeuille dont il dispose et l'identifiant unique qui y est associé.

Vous devez donc ici créer une nouvelle classe "Wallet", dans laquelle vous pouvez dors-et-déjà intégrer un attribut "unique_id", qui comportera l'identifiant unique d'un utilisateur.

2.5 Mise en place de la classe de gestion des blocs

Comme expliqué ci-avant, la blockchain est une succession de blocs. Vous aurez donc besoin d'une classe "Block" qui vous permettra de gérer l'ensemble des blocs. Créez cette classe.

3 Mise en place de la persistance des données

3.1 Principes d'une base de données partagée et décentralisée

Le principe fondamental de sauvegarde des données d'une blockchain réside dans le fait qu'elle soit accessible, dupliquée, et connue de tous ses utilisateurs. Ainsi la blockchain EST la base de données, en plus de tous les autres outils de développement que vous aurez à mettre en place.

Cette définition induit donc le fait que toute personne prenant part à une blockchain - de par la création d'un wallet - dispose non seulement dudit wallet, mais également de l'ensemble des informations relatives à la blockchain - à savoir la liste des wallets existants, la liste de l'ensemble des blocs existants, etc…

Ne pouvant bien évidemment pas recréer, en 2ème année et en 2 jours, une blockchain complète, nous allons ici sauvegarder l'ensemble des informations dans le dossier "content" que vous avez créé lors des étapes précédentes.

3.2 Organisation du dossier "content"

Le dossier "content" servira donc à stocker l'ensemble des informations de vos blockchain, et constitue donc votre base de données. Celui-ci va être séparé en plusieurs sous-dossiers, qui contiendront eux-même des fichiers JSON. Ces derniers serviront donc à sauvegarder vos données en les inscrivant dedans.

Vous allez ici devoir créer plusieurs sous-dossiers, parmis lesquels :

- un dossier "**wallets**" - qui servira à stocker l'ensemble des wallets et des informations afférentes
- un dossier "**blocs**" - qui contiendra l'ensemble des informations contenus dans les blocs

4 Implémentation des utilisateurs

4.1 Génération des identifiants uniques des wallets

Les utilisateurs sont donc représentés par des "wallets" au sein d'une blockchain. Ainsi chaque fois qu'une personne crée un "wallet", un nouvel utilisateur est réputé créé. Nous avons dors-et-déjà créé la classe Wallet qui nous servira donc pour la suite de cette partie.

Comme vu précédemment, chaque wallet dispose d'un identifiant unique. Vous devez donc ici créer une méthode permettant de générer un identifiant unique. Vous pouvez appeler cette méthode "generate_unique_id()", et vous devez l'implémenter de telle sorte à ce qu'elle retourne un identifiant unique. Pour ce faire, vous devrez vous servir de la librairie "uuid" qui vous permettra de parvenir à vos fins.

NB : Vous devrez par la suite ajouter des vérifications dans cette méthode de telle sorte à être certain que l'identifiant généré ne soit pas déjà utilisé par un autre wallet.

4.2 Ajout des informations propres aux wallets

Les portefeuilles des blockchains comprennent de nombreuses informations. Pour ce TP, nous nous contenterons de quelques-unes :

- L'identifiant unique, que vous avez normalement déjà ajouté
- Le solde du portefeuille. Vous pouvez donc créer un attribut "balance" qui contiendra un nombre
- L'historique des transactions. Vous pouvez ici créer un attribut "history" qui contiendra un dictionnaire ou une liste.

Vous devrez ensuite implémenter plusieurs méthodes qui vous serviront par la suite :

- Une méthode pour ajouter un montant à l'attribut "balance". Vous nommerez cette méthode "add_balance()" et la rendrez fonctionnelle.
- Une méthode pour déduire un montant de l'attribut "balance". Vous nommerez cette méthode "sub_balance()" et la rendrez fonctionnelle.
- Une méthode pour ajouter une transaction dans l'historique. Appelez cette méthode "send()". Vous développerez le contenu de cette méthode plus tard. Vous pouvez simplement mettre "pass" dedans pour le moment.

4.3 Mise en place de la sauvegarde des wallets

La sauvegarde des wallets devra se faire au format JSON. Pour la réaliser, vous devrez donc convertir votre objet "Wallet" au format JSON, puis l'enregistrer dans un fichier. Voici quelques indications à suivre pour cette étape :

- Le fichier devra être sauvegardé dans le dossier "content/wallets"
- Le nom du fichier devra correspondre à l'identifiant unique du wallet en question
- Vous devrez intégrer l'ensemble des informations du wallet dans le fichier JSON en question
- Veillez à bien enregistrer le fichier avec l'extension JSON

La réalisation de l'ensemble de ces étapes devra être faite dans une méthode nommée "save()".

4.4 Mise en place de la récupération des informations des wallets

Le chargement d'un objet Wallet à partir d'un fichier JSON doit également être possible. Vous devez pour se faire ouvrir le fichier correspondant à l'identifiant unique du wallet souhaité, charger les données du JSON et les intégrer dans les attributs d'un objet Wallet.

Vous réaliserez cette étape grâce à une méthode nommée "load()".

4.5 Finalisation de la génération des identifiants uniques

Revenez sur la méthode de génération des identifiants uniques, et ajoutez des vérifications de telle sorte à ce qu'il ne puisse pas exister 2 fois un Wallet disposant du même identifiant

5 Implémentation des blocs

Le fonctionnement des blocs constitue le socle de la blockchain, il est donc important de bien comprendre les bases de leurs fonctionnement avant d'aller plus loin.

5.1 Fonctionnement simplifié des blocs

La dénomination "bloc" sert davantage à représenter humainement parlant le concept de blockchain, plutôt que leurs fonctions. Un bloc est constitué de plusieurs éléments, parmis lesquels :

- une taille
- un nom
- un parent
- une liste de transactions

La **taille** du bloc constitue la limite de contenu que peut accueillir ledit bloc. Afin de mettre en place une limite au sein de ce projet, vous allez devoir faire en sorte que chacun des blocs n'excède pas 256 000 octets. La sauvegarde de ceux-ci s'effectuant au format JSON pour ce projet - de la même manière que les Wallets - il vous faudra être capable de récupérer le poids d'un fichier.

Le **nom** du bloc est constitué de ce qu'on appelle un "**hash**". En cryptographie, un hash est le résultat d'une fonction mathématique appliquée à un objet. La principale différence entre un "hash" et un "cryptage" réside dans le fait que le cryptage est conçu pour pouvoir être décrypté, alors que le hash ne l'est pas. Par ailleurs, la particularité des fonctions de "hashage" est qu'un objet sera toujours "hashé" de la même manière. Ainsi par exemple : la lettre "a", une fois hashé en SHA256, deviendra systématiquement "ca978112ca1bbdcafac231b39a23dc4da786eff8147c4e72b9807785afee48bb" et ne pourra jamais donner de résultat différent.

Le **parent** du bloc représente le bloc auquel est rattaché un nouveau bloc. Souvenez vous qu'une "blockchain" est une "chaîne de blocs". Aussi tous les blocs sont rattachés entre eux ! Lorsqu'un nouveau bloc est créé, on lui affecte un bloc parent afin qu'il puisse s'intégrer à la chaîne dans sa globalité.

La **liste des transactions** est, comme son nom l'indique, une liste de l'ensemble des transactions qui sont enregistrées sur un bloc. Ainsi pour qu'une transaction soit effectuée sur une blockchain, il faut impérativement qu'un bloc dispose de suffisamment de place afin d'inscrire cette transaction (en fonction de sa **taille**).

NB : Les parties suivantes vous détailleront comment mettre en place un fonctionnement basique de génération de blocs. N'ayez pas peur, vous serez guidé dans la réalisation de ces étapes. A titre informatif, vous devez ici relever qu'il faut donc des "blocs" pour inscrire des

transactions dessus, et que ces blocs sont nommés par des “hash”. Le fait de trouver un nouveau “hash” (en fonction de critères définis) induit la création d’un bloc. En réalité un bloc est créé à chaque nouveau “hash” trouvé. Le fait de réaliser des calculs pour trouver de nouveaux hash est le fondement même de la preuve de travail - ou PoW : Proof of Work.

5.2 Mise en place des attributs

Voici la liste des attributs dont vous aurez besoin au minimum :

- **base_hash** : nombre de base que vous avez utilisez afin d’obtenir votre “hash”.
- **hash** : résultat du hashage de votre chaîne de caractère
- **parent_hash** : hash du bloc parent
- **transactions** : liste des transactions réalisées et enregistrées au sein du bloc

5.3 Implémentation des méthodes

Plusieurs méthodes sont ici à implémenter pour la classe “Block” :

- **check_hash()** : méthode permettant de vérifier que le “**hash**” du bloc est correct en fonction du “**base_hash**”.
- **add_transaction()** : Méthode permettant d’ajouter une transaction au bloc. Une transaction concerne un wallet émetteur, un wallet récepteur et doit également contenir un montant.
- **get_transaction()** : Méthode permettant de récupérer une transaction par rapport à son numéro (CF point “transactions” ci-dessous).
- **get_weight()** : Méthode permettant de récupérer le poids total du bloc.
- **save()** : Similaire à la méthode homonyme des Wallets, l’identifiant unique ici étant l’attribut “hash”.
- **load()** : Similaire à la méthode homonyme des Wallets, l’identifiant unique ici étant l’attribut “hash”.

NB : L’algorithme de hashage utilisé pour ce projet devra être le SHA256. Vous pouvez accéder à celle-ci grâce à la librairie “hashlib” disponible en python.

Vous devez donc être en capacité de gérer des portefeuilles ainsi que des blocs. Il est temps d’implémenter la gestion de la chaîne dans son ensemble !

6 Implémentation de la chaîne

6.1 mise en place des attributs

Voici la liste des attributs dont vous aurez besoin pour gérer votre blockchain :

- **blocks** : Liste des blocs au sein de la chaîne.
- **last_transaction_number** : Dernier numéro de transaction enregistré.

6.2 mise en place des méthodes

Voici une liste des méthodes que vous devrez implémenter :

- **generate_hash()** : Méthode qui vous permettra de créer des blocs. Pour rappel, la création d'un bloc s'effectue par la découverte d'un nouveau hash en fonction de certains critères. Voici la méthode que vous devrez utiliser afin de chercher un nouveau hash :

```
Import hashlib
hash = hashlib.sha256("a".encode()).hexdigest()
```

Dans l'exemple ci-dessus, c'est la lettre "a" qui est hashée, c'est donc celle-ci que vous devrez remplacer par une autre chaîne de caractère afin de déterminer si vous remplissez les conditions pour trouver un nouveau bloc (cf méthode suivante).

- **verify_hash()** : Méthode permettant de vérifier si un hash correspond aux critères de la chaîne ou non. Les critères sont les suivants :
    - le hash ne doit pas déjà exister
    - le hash doit systématiquement commencer par quatre "zéro" (0000).

Pour reprendre l'exemple du caractère "a", les 4 premiers caractères du hash sont "ca97". Ils ne remplissent donc pas les conditions pour permettre de créer un bloc.

!!! Important !!! : Vous devrez implémenter une méthode de génération de chaînes de caractères afin de pouvoir générer des hashs et en trouver qui correspondent aux critères ci-dessus. Vous êtes libre du nom de la méthode et de son implémentation. Pour plus de simplicité, vous pouvez simplement utiliser des nombres et les incrémenter dans une boucle.

- **add_block()** : Méthode permettant de créer un nouveau bloc. Une fois un nouveau hash trouvé, vous devrez utiliser cette méthode afin de créer un bloc, et donc enregistrer un nouveau fichier JSON grâce à la méthode "save()" déjà créée. Vérifiez bien que le hash en question ne soit pas déjà existant (et donc qu'un bloc homonyme n'existe pas déjà).
- **get_block()** : Méthode permettant de récupérer un bloc en fonction de son hash
- **add_transaction()** : Méthode permettant d'ajouter une transaction à un bloc. Cette méthode appellera la méthode homonyme du bloc dans lequel elle sera enregistrée.

NB : Le premier bloc de la chaîne ne peut pas contenir de bloc parent. De la sorte, vous pourrez ici seulement créer ce premier bloc à la main, en le nommant "00" manuellement, et en définissant le bloc parent à "00" manuellement également. Ce bloc sera donc votre socle pour la création des suivants.

7 Implémentation des transactions

Les transactions doivent être implémentées - pour ce TP - dans la classe "Chain". Vous êtes libre de l'implémentation, cependant gardez en tête l'ensemble des éléments suivants :

- Pour valider une transaction, vous devez vérifier l'ensemble des éléments suivants :
    - Le Wallet de l'émetteur existe
    - Le Wallet du récepteur existe
    - Le solde du wallet émetteur est suffisant pour réaliser la transaction
- Pour procéder à une transaction, vous devez vérifier qu'il est possible d'écrire une transaction sur un bloc
- Vous devez obligatoirement écrire la transaction dans un bloc. L'écriture doit changer le fichier contenant le bloc en conséquence !

8 Préparation préliminaire du rendu

Maintenant que votre chaîne est fonctionnelle, il est temps de la tester par la pratique. Voici quelques étapes supplémentaires afin d'y parvenir :

- Lors de la création d'un wallet, vous devez automatiquement attribuer 100 tokens au nouvel utilisateur. Ce n'est pas le fonctionnement normal de la blockchain comme vous vous en doutez, cependant cela vous permettra de faire vos tests par la suite.
- Dans le dossier "settings", créez un fichier "settings.py", puis définissez-y cette constante :
    - **MAX_TOKEN_NUMBER** : défini le nombre maximum de token existant sur la chaîne. Vous devrez ajouter des vérifications pour que des tokens ne puissent pas être créés par des problèmes de calculs
- Il vous reste ici 2 méthodes à mettre en place afin de pouvoir commencer à jouer avec votre blockchain :
    - **find_transaction()** : méthode contenu dans la classe "Chain", qui doit retourner l'objet "Block" dans lequel se trouve le numéro de transaction demandé en paramètre.
    - **get_last_transaction_number()** : méthode contenu dans la classe "Chain", qui doit retourner le dernier numéro de transaction réalisé. La première transaction doit être la numéro 1, la seconde : 2, etc… Cette méthode vous permettra d'y parvenir. Pensez bien à mettre à jour la création des transactions en conséquence.

9 Finalisation du rendu

Maintenant que votre blockchain est fonctionnelle, il est temps de préparer un contenu afin de prouver l'efficacité de votre travail ! Pour cela, servez-vous du fichier "main.py", normalement à la racine de votre dossier, et placez-y l'ensemble des éléments vous permettant de valider les points de la partie "fonctionnalités" du barème d'évaluation ci-dessous. N'hésitez pas à en mettre "trop", plus vous êtes capable de démontrer de fonctionnalités, mieux vous serez noté !

10 Barème d'évaluation

- **(total 4 pts)** common skills :
    - **(1pt)** mail de rendu professionnel
    - **(1pt)** lien du repository git accessible
    - **(1pt)** commits réalisés régulièrement
    - **(1pt)** readme présent et détaillant les étapes d'installation et d'utilisation de votre programme
    - **(-2 pts)** travail rendu en retard
- **(total 15 pts)** respect des consignes :
    - **(3 pts)** l'architecture des dossiers et fichiers est respectée
    - **(5 pts)** les noms des attributs et méthodes imposés sont respectés (-1 pt par erreur)
    - **(5 pts)** flake8 ne retourne aucune erreur
    - **(2 pts)** les librairies "uuid" et "hashlib" sont utilisées (1pt par librairie)
- **(total 61 pts)** fonctionnalités :
    - **(sous-total 11,5 pts)** la classe Wallet est complète et fonctionnelle
        - **(0,5 pt)** attribut "unique_id"
        - **(0,5 pt)** attribut "balance"
        - **(0,5 pt)** attribut "history"
        - **(2 pts)** méthode "generate_unique_id"
        - **(1 pt)** méthode "add_balance"
        - **(1 pt)** méthode "sub_balance"
        - **(3 pts)** méthode "save"
        - **(3 pts)** méthode "load"
    - **(sous-total 16 pts)**la classe Block est complète et fonctionnelle
        - **(0,5 pt)** attribut "unique_id"
        - **(0,5 pt)** attribut "hash"
        - **(0,5 pt)** attribut "parent_hash"
        - **(0,5 pt)** attribut "transactions"
        - **(1 pt)** méthode "check_hash"
        - **(2 pts)** méthode "add_transaction"
        - **(4 pts)** méthode "get_transaction"
        - **(1 pt)** méthode "get_weight"
        - **(3 pts)** méthode "save"
        - **(3 pts)** méthode "load"
    - **(sous-total 12,5 pts)**la classe Chain est complète et fonctionnelle
        - **(0,5 pt)** attribut "blocs"
        - **(0,5 pt)** attribut "last_transaction_number"
        - **(5 pts)** méthode "generate_hash"
        - **(0,5 pts)** méthode "verify_hash"
        - **(1 pt)** méthode "add_block"
        - **(2 pts)** méthode "get_block"
        - **(3 pts)** méthode "add_transaction"
    - **(sous-total 21 pts)** les transactions sont fonctionnelles
        - **(5 pts)** vérification de l'existence du wallet émetteur
        - **(5 pts)** vérification de l'existence du wallet récepteur
        - **(2 pts)** vérification du solde du wallet émetteur
        - **(3 pts)** vérification de la disponibilité d'un bloc pour écriture
        - **(3 pts)** changement du contenu du bloc dans le fichier avec nouvelle écriture

- **(3 pts)** changement du solde de l’émetteur et du récepteur en conséquence

11 Pour aller plus loin…

Vous aurez compris - si vous êtes arrivé au bout de ce TP - que vous n’avez mis ici en place que les rudiments d’une blockchain. Afin d’aller plus loin, vous pouvez vous renseigner sur les principes de validations de transactions et de preuve de travail. Ajoutez à cela un moyen de partage efficace (autre que GIT ^^) des informations, et vous devriez commencer à entrevoir un semblant de blockchain réaliste, auquel il vous manquera une bonne dose de sécurisation !