Le processus de demande de crédit à la consommation

Master Code Camp

Crypto4All - Défi 1

Fonctionnement du prêt à la consommation

- Le crédit à la consommation est proposé par les banques et les sociétés financières.
- Le client doit fournir son état civil, ses revenus et ses charges ainsi que le montant souhaité et la durée.
- La banque vérifie son taux d'endettement et lui accorde ou non un crédit.

Processus actuel

- Aujourd'hui pour faire un crédit à la consommation on contacte soit une banque soit une société financière.
- L'organisme créé un dossier qui va être traité par un salarié.
- Le client va apporter toutes les pièces pour monter son dossier.
- Le salarié va valider ou invalider le dossier.
- Le client recevra (ou non) l'argent souhaité.

Notre proposition

- Automatiser la demande de crédit à la consommation avec l'utilisation de la blockchain.

Les coûts

Actuel	Notre solution
Frais postaux 1,70€ pour 15 feuilles A4	Ethereum 0€ -> blockchain privée
Les coûts sont estimés sur une base de 30 min de traitement en agence ou par téléphone au coût du smic (7,58€/h) soit 3,79€.	
Les coûts sont estimés sur une base de 30 min de traitement en service administratif bancaire au coût du smic (7,58€/h) soit 3,79€.	

Coûts de fonctionnement de l'infrastructure équivalents.

Lean Canvas

Problem

Le processus de demande de prêt demande une intervention humaine.

Solution

A l'aide de smart contracts nous pouvons supprimer l'intervention humaine dans ce processus et externaliser le stockage des données, et ainsi réaliser des économies.

Key Metrics

Il y a près de 7,4 millions de ménages ayant contracté un nouveau crédit en 2016.

Unique Value Proposition

Automatiser le processus de demande de prêt à la consommation grâce à la blockchain.

High level concept

« Blockchainiser » les prêts à la consommation.

Unfair advantage

Utilisation de la blockchain, tout le processus de validation sera sécurisé.

Channels

Interface web de la banque, depuis le compte utilisateur client.

Customer segment

Les banques et les institutions de crédit.

Cost structure

La banque peut faire payer pour se service tout comme elle fait payer pour tous ces autres services. L'accès à l'interface web de la banque est payant, elle peut faire demander des frais pour rembourser les coûts de traitements sur le réseau ethereum qui devraient être minimes par rapport au coût d'un employé qui va gérer un dossier.

Revenue Streams

Nous allons faire payer le logiciel à la banque pour qu'ils puissent l'ajouter dans leurs interfaces web.

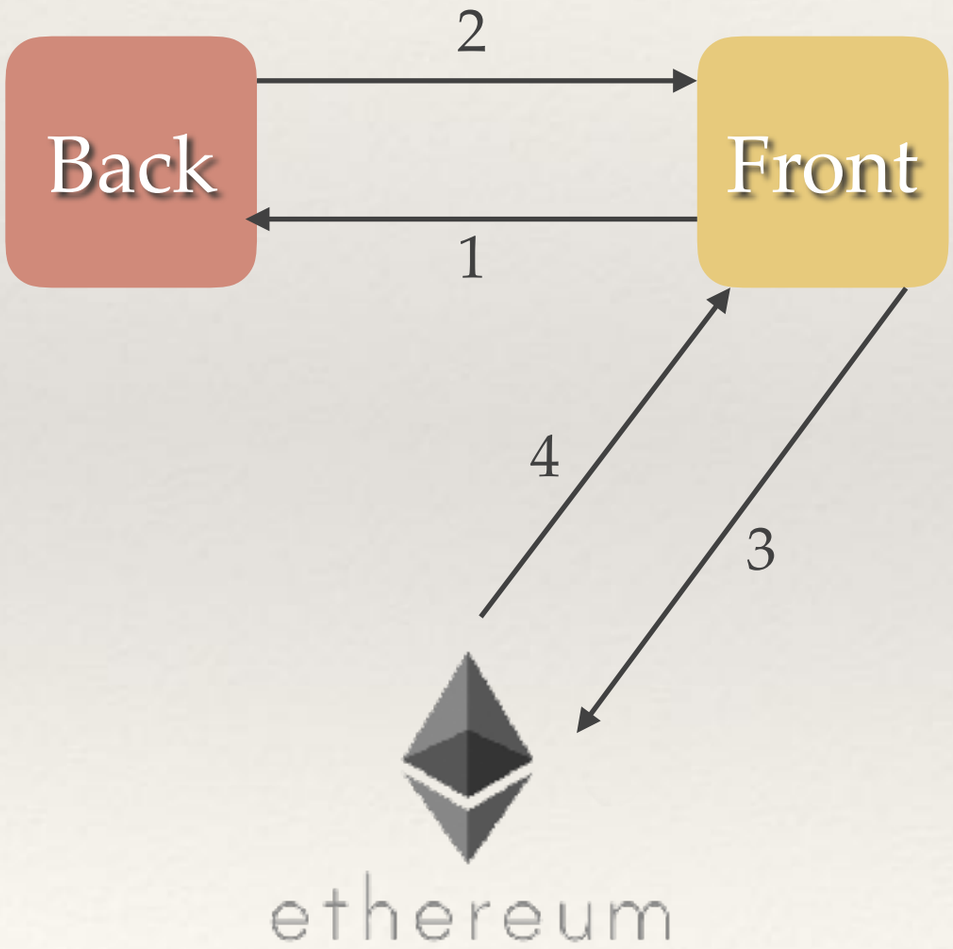
Partie Technique

Architecture technique

- Les technologies que nous avons choisies sont : Geth, Truffle, TestRPC (JSON-RPC) sur le réseau Ethereum et Solidity.
- Nous avons une blockchain privée avec deux noeuds et un smart contract.
- Nous avons fait une interface front end en ReactJS pour faire la demande de crédit (celle-ci sera optionnelle en fonction des besoins client).

Fonctionnement / Interactions

Schéma des interactions



Légende :

1. Demande de validation
2. Réponse avec gas et adresse du noeud contenant le smart contract
3. Appel au smart contract pour valider le prêt
4. Réponse du smart contract

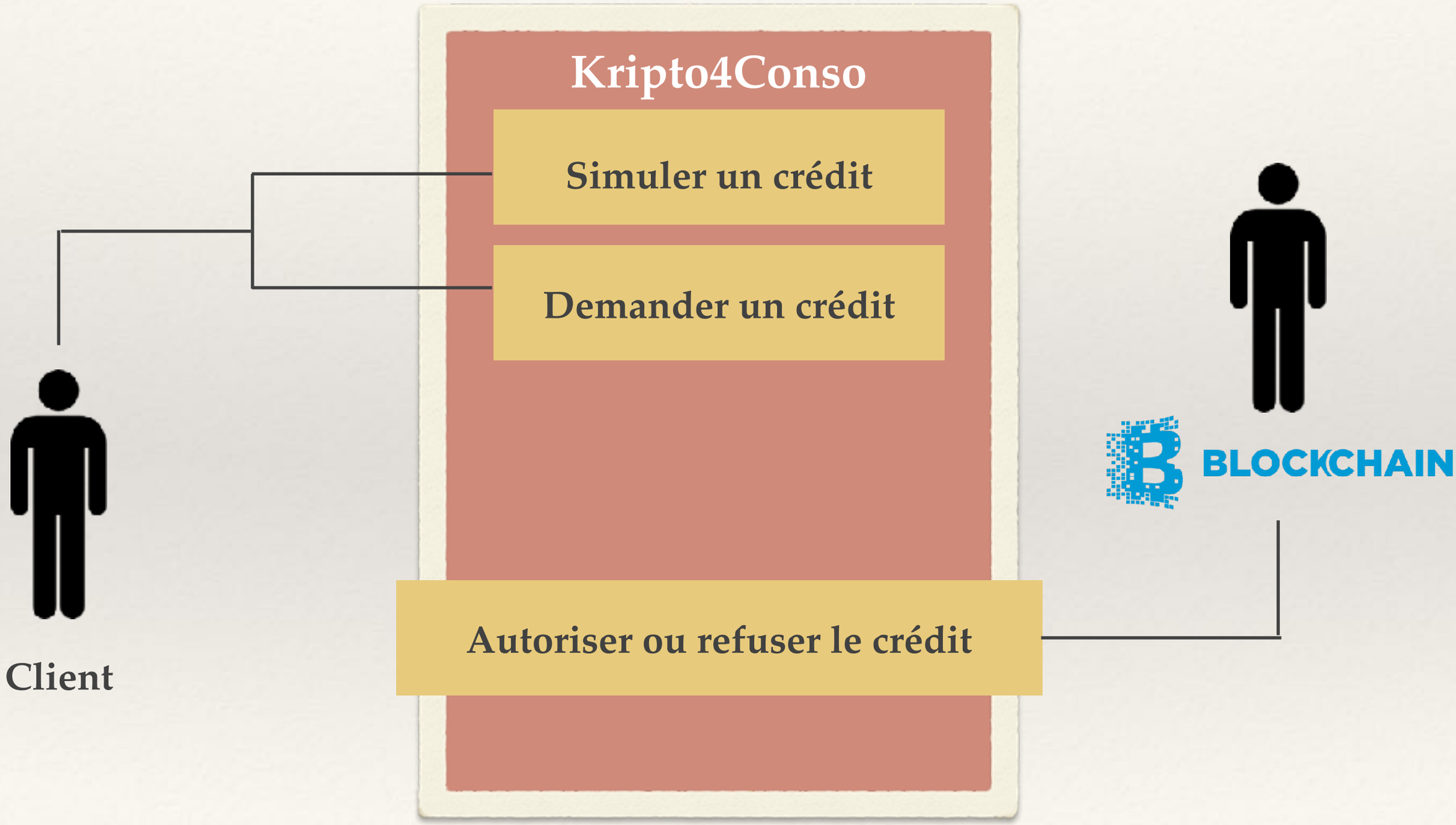
Smart Contract

- Avantages :
  - Données infalsifiables
  - Réseau sécurisé
  - Réduction des coûts
  - Satisfaction client
- Problématiques :
  - Problème de sécurité
  - Problème de gestion de l'ethereum
  - Problématique d'environnement (doc/lib/etc…)

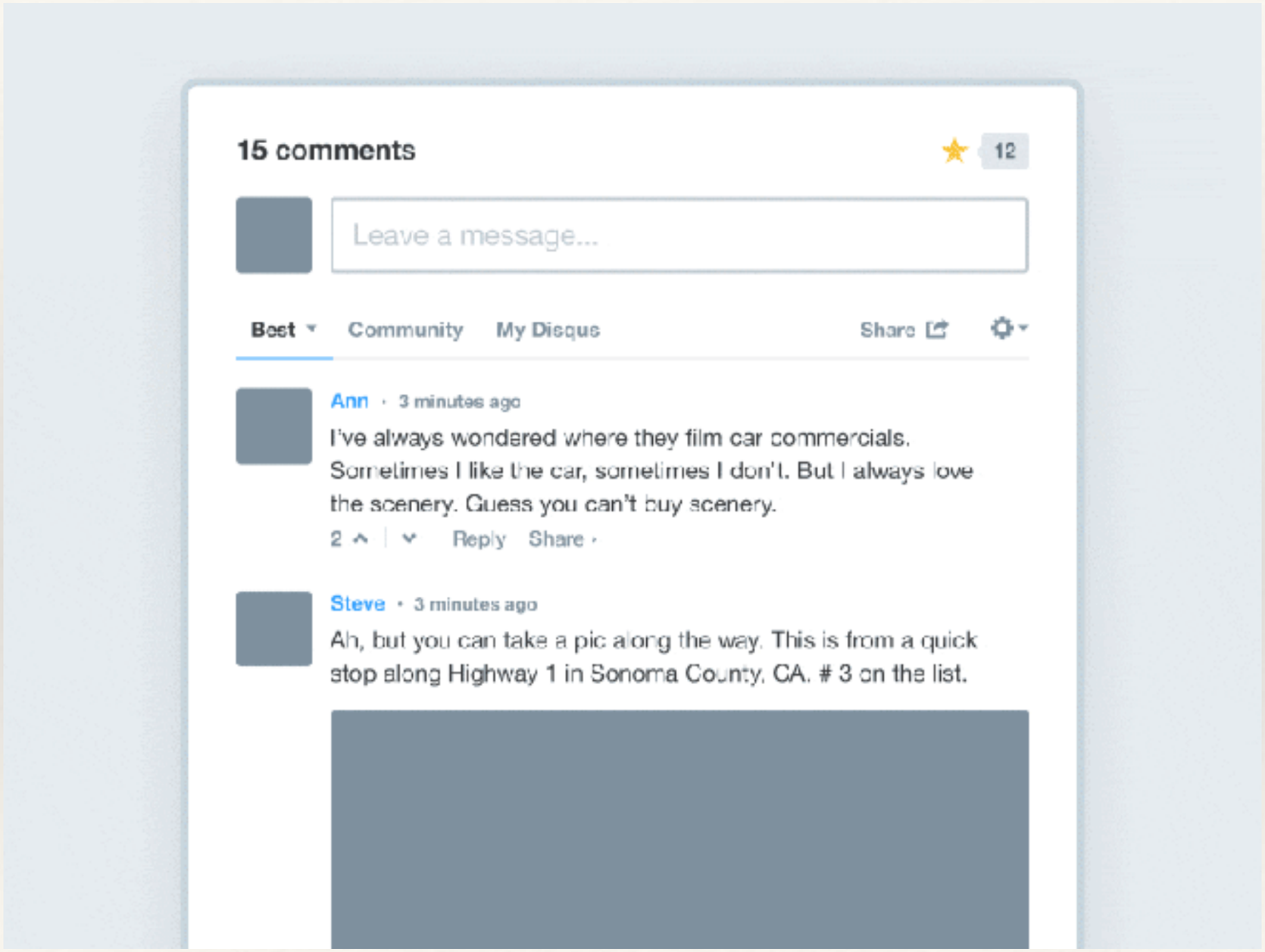
Sécurité / Gestion ethereum

- Problématiques :
  - Empêcher de se faire miner de l'ether.
  - Empêcher de faire des appels inutiles.
  - Documentation, librairies et tiers, utilisation.

Diagramme de cas d'utilisation



Démonstration de notre application



Ethereum GO

```
{
"nonce": "0x00",
"difficulty": "0x1000",
"mixHash": "0x00",
"timestamp": "0x00",
"parentHash": "0x00",
"extraData": "0x00",
"gasLimit": "0x1000000000"
}
```

- **nonce** : hash que le mineur fait varier pour résoudre le proof-of-work
- **difficulty** : difficulté de la preuve de travail
- **mixmash** : hash de l'entête du bloc sur lequel se base le mineur pour trouver le nonce
- **timestamp** : moment de validation du bloc
- **parentHash** : hash du bloc précédant
- **extraData** : argument facultatif pour stocker des données (32 octets max.)
- **gasLimit** : montant maximum de *gas* que les contrats peuvent consommer

Notre blockchain

```
$ geth --datadir ./noeud1 --networkid "100" init genesis.json
```

```
$ geth --datadir ./noeud2 --networkid "100" init genesis.json
```

```
$ geth --datadir ./noeud1 --networkid "100" account new
```

```
$ geth --datadir ./noeud2 --networkid "100" account new
```

```
Your new account is locked with a password. Please give a password. Do not forget this password.
Passphrase:
Repeat passphrase:

Address: {ed3b402b69a4ca6428c2a0cee8adb6e82765c220}
```

Notre smart contract

```
contract Validation {

    struct Client {
        address from;
        bool used;
    }

    struct ClientValidation {
        address from;
        bool accepted;
    }

    mapping(address => Client) public tab_client;
    mapping(address => mapping(uint => ClientValidation)) public map;

    function setClientValidation(address client, bool accepted) internal {
        Validation.map[client][block.timestamp] = ClientValidation(client, accepted);
    }

    function giveEther(address client) returns (bool) {
        if (Validation.isValid(client) == false) {
            return false;
        }

        Validation.tab_client[client] = Client(client, true);
        return true;
    }


    function isValid(address client) returns (bool) {
        if ((Validation.tab_client[client].from != 0) && Validation.tab_client[client].used == true) {
            return false;
        }

        return true;
    }

    function calculMounthly(uint amount, uint duration, uint rate) internal returns (uint) {
        return ((amount * rate / 12) / (1 - (1 + rate / 12) ** -duration));
    }

    function calculContractIntern(uint pay, uint accruedLiabilities, uint monthly) internal returns (bool) {
        uint indebtedness = (accruedLiabilities + monthly) / (pay / 12);
        if (indebtedness > 32) {
            return (false);
        }

        return (true);
    }

    function checkIfContractValid(address client, uint amount, uint duration, uint rate, uint pay, uint accruedLiabilities) returns (bool) {
        if (Validation.tab_client[client].from != 0) {
            Validation.tab_client[client].used = false;
        }

        uint monthly = calculMounthly(amount, duration, rate);

        if (calculContractIntern(pay, accruedLiabilities, monthly) == false) {
            Validation.setClientValidation(client, false);
            return false;
        }

        Validation.setClientValidation(client, true);
        return true;
    }
```

Infos projet

- Communication Slack
- Dépot git : https://github.com/misseur/Kriptonite4all

Merci pour votre attention