EZ CREDIT

ABSCHLUSSBERICHT

BLOCKCHAIN HS 18

Brandenberg Nicola, Kiser Kimia, Scheller Rico

INHALTSVERZEICHNIS

BUSINESS CASE

Im Folgenden wird der Businessplan des fiktiven Unternehmens EZCredit vorgestellt. Es handelt sich dabei um eine internationale, auf Blockchain basierte Micro Lending Plattform. Die Problemstellung und das Innovationspotential werden vorgestellt und der Einsatz einer dezentralen Lösung wird vorgestellt.

1. GRÜNDUNGSMITGLIEDER

Chief Financial Officer, CFO	Brandenberg Nicola
Chief Executive Officer, CEO	Kiser Kimia
Chief Technology Officer, CTO	Scheller Rico

2. IDEE

Wenn man heutzutage einen Kredit aufnehmen will, ist man von den Kreditinstituten abhängig. Erstens bewerten sie den potentiellen Kunden undurchsichtig. Da die Kreditangaben von den diversen Kreditkartenanbietern und sonstigen Kreditinstituten anderen Teilnehmern mitgeteilt werden, führt eine Falscheinschätzung eines Unternehmens zu weitreichenden Nachteilen für den potentiellen Kunden. Zweitens diktieren die Kreditinstitute den Zins, welcher an das Darlehen gekoppelt ist. Dadurch ist es für Teile der Bevölkerung aufgrund von möglicher Fehleinschätzung oder sogar Diskriminierung nicht möglich einen Kredit zu erhalten.

Länder mit hoher Inflation und schwachen Währungen sind ebenfalls betroffen. In Ghana beläuft sich das Durchschnittliche Monatseinkommen auf 1450 GHS[1], was umgerechnet rund 343 CHF beträgt. Mit einer Leihe von nur 100 CHF könnte man also das Einkommen um fast einen Drittel erweitern.

EZCredit verbessert diese Situation indem das Unternehmen sich ganz speziell auf Leihen von kleiner als 1000 CHF spezialisiert, also Micro Lending betreibt. Zusätzlich kann bei einem Kreditantrag auch der gewünschte Zinssatz festlegt werden. So kann ein Kreditnehmer individuell den Zins festlegen, den er zu zahlen gewillt ist. Dies schafft ein einfaches und flexibles System, das sich an die jeweilige Situation anpassen lässt. Alternativ kann auch ein Kreditantrag ohne Zins aufgegeben werden, worauf sich Anbieter beim Kunden mit ihrem Angebot melden können. Der Kunde kann dann das Angebot mit dem besten Zins auswählen.

Um die Glaubwürdigkeit eines Kunden einzuschätzen wird ein intern entwickeltes Prüfsystem verwendet. Anhand von gelungenen und misslungenen Rückzahlungen kann so auf die Kreditwürdigkeit geschlossen werden.

3. BEGRÜNDUNG EINSATZ DER BLOCKCHAIN

Da die potentiellen Kreditnehmer den Kreditgebern nicht vertrauen, die Kreditwürdigkeit korrekt zu bewerten und alle ihre bisherigen erfolgreichen Rückzahlungen auch an die anderen Kreditgeber zu propagieren, ist die Nutzung einer Blockchain nötig. In der Blockchain kann jeder den Code und somit den Bewertungsvorgang einsehen. Die Smart Contracts garantieren des Weiteren die korrekte Ausführung. Eine Blockchain gibt also den Benutzern die nötige Sicherheit, um dem Prozess zu vertrauen.

[1] (vergleiche https://de.tradingeconomics.com/ghana/wages-high-skilled)

Wie in Abbildung 1 zu sehen ist, bildet ein Smart Contract (grün) die zentrale Stelle, durch welche beide Parteien ihre Zahlungen abwickeln. Dadurch kann jeder einsehen, ob Person A jemals einen Kredit bekommen hat, welchen sie nicht zurückgezahlt hat.

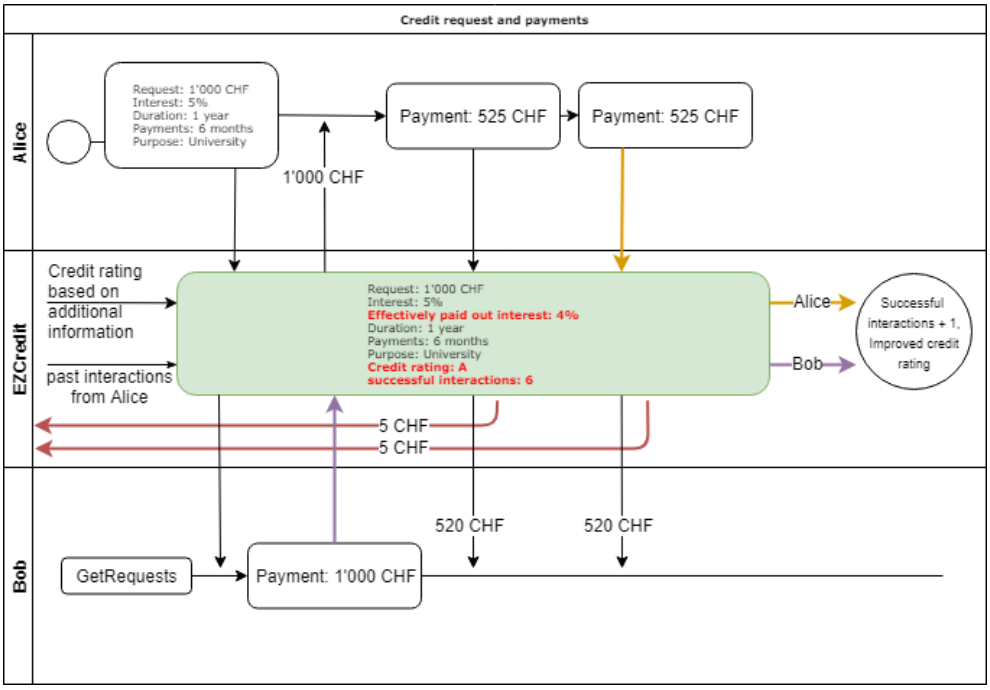

Abbildung 1: Credit Request and Payments

4. WIRTSCHAFTLICHKEIT

EZCredit finanziert sich über eine Provision welche bei jeder erfolgreichen Kreditvermittlung erhoben wird. Um rentabel zu sein wird mit einer Abgabe ca. 20% der anfallenden Zinsen gerechnet.

5. KONKURRENZ UND ABSATZMARKTANALYSE

Der Markt für Kredite ist in den letzten Jahren stark angestiegen. Dabei ist ersichtlich, dass immer mehr Personen ihren Kredit nicht bei einer normalen Bank aufnehmen, sondern über einen Anbieter im Internet.

Klassische Konkurrenten sind Banken wie Migros Bank, Cembra, Raiffeisen usw., welche Bankkredite mit Zinsen von ca. 6-10% vergeben. Weitere direkte Konkurrenten sind Unternehmen wie zum Beispiel "Lend.ch", welche ein Crowdlending, sprich ein Peer-to-Peer Kredit anbieten. Dabei werden die Kredite durch Privat-Anleger finanziert, welche für ihre Geldanlagen einen festen Zins erhalten.

Was EZCredit jedoch grundlegen von bisherigen Kreditinstituten unterscheidet ist der Fakt, das EZCredit auf einer Blockchain aufgebaut ist und sich nur auf Kleinkredite spezialisiert. Dies ermöglicht eine anonyme und verlässliche Datenverwaltung. Die Prüfung der Kreditwürdigkeit ist global unabhängig und nicht von Politik, Kultur, Staaten oder Religion beeinflusst. Zugleich verhindert die Blockchain durch eine gesicherte Authentifizierung die Gefahr eines Identitäten Diebstahls und somit eines Kreditmissbrauchs.

6. SWOT-ANALYSE

Unternehmensanalyse (interne Analyse)

Stärken:
- Innovationspotential
- Wettbewerbsfähigkeit
- Produkt- und Dienstleistungsqualität
- Know-how / qualifiziertes Personal

Schwächen:
- Liquidität
- Kostenstruktur
- An Währung gebunden (ETH)
- Abhängigkeit von Konkurrenz

Umweltanalyse (externe Analyse)

Chancen:
- Neuartiges Geschäftsfeld / Technologie
- Marktwachstum im Krypto Bereich
- Vertrauensverlust in Kreditinstitute

Gefahren:
- Neuartiges Geschäftsfeld, daher noch kaum durchdrungen Gesetzgebungen
- Politische Entwicklungen und Konjunkturschwankungen

SWOT-Matrix

	Stärken (Strenghts)	Schwächen (Weaknesses)
Chancen (Opportunities)	• Innovationspotential nutzen, um neue Lösung auszubauen • Mit Know-how das neue Geschäftsfeld / Technologie erweitern • Wettbewerbsfähigkeit im Hinblick auf einen stark wachsenden Markt	• Durch Marktwachstum auch steigende Anzahl Investoren • Expandierender Markt von Krypto Währungen führt zu mehr Nutzern / Besitz von Ether
Gefahren (Threats)	• Neuartigkeit des Geschäftsfeldes durch Innovationspotential meistern • Durch qualifiziertes Personal und Know-how sich mit Gesetzgebungen auseinandersetzen	• Laufend Investoren an Bord holen • Positionierung kontinuierlich anpassen • Umwelt und Trends verfolgen und Strategie dementsprechend adaptieren

ARCHITEKTUR

Im Folgenden wird die Systemarchitektur des fiktiven Unternehmens EZCredit vorgestellt. Die Technische Anforderung an die Lösung wird besprochen, die Wahl des Protokolls und Consensus Algorithmus wird begründet und das Zusammenspiel der Komponenten wird mit Grafiken unterstützt.

1. TECHNISCHE ANFORDERUNG AN DIE LÖSUNG

Es wurde diskutiert, ob für jede Kreditanfrage ein neuer Smart Contract aufgesetzt oder alles in einem Contract abgehandelt werden soll. Nur einen Contract zu verwenden und die Daten (wer, wieviel, an welche Adresse etc.) off-Chain zu speichern, würde wahrscheinlich günstiger ausfallen. Dies würde aber die Transparenz der Blockchain untermauern, weil dann die Benutzer wieder Vertrauen haben müssten, dass die Daten richtig sind und nicht manipuliert werden.

Ausserdem geben alleinstehende Contracts die Chance, dass Leute auch nicht über unser GUI einbezahlen müssen, sondern auf ihre bevorzugte Art und Weise die Zahlungen machen können, weil die Smart Contracts die nötige Logik implementiert haben.

Zudem war im Rahmen dieser Arbeit auch einfach interessant zu sehen, wie ein Smart Contract einen anderen Contract erzeugen kann.

2. PROTOKOLL UND CONSENSUS ALGORITHMUS: ETHEREUM

EZCredits verwendet zur Implementierung der Blockchain das Ethereum-Netzwerk. Ethereum ist eines der ersten Blockchain-Protokolle, welches neben Finanztransaktionen auch komplexe Logik ausführen kann. Dazu bietet Ethereum ein ausgereiftes Konzept mit Smart Contracts, welche für die Funktionsweise der Kredit-Vergabe zwischen unbekannten Partner von elementarer Bedeutung ist.

Ein entscheidender Vorteil von Ethereum im Vergleich zu beispielsweise Bitcoin ist die Möglichkeit zur Entwicklung von Dapps (Destributed Apps). Des Weiteren erlaubt es Ethereum neue Währungen zu entwickeln, was eine starke Abhängigkeit von einer grossen Kryptowährung verhindert.

3. BESTANDTEILE DER LÖSUNG: ORACLES

Da ein Kredit in Ether den immensen Kursschwankungen einer Kryptowährung im momentanen Umfeld ausgesetzt würde, könnte kein Benutzer sicher sein, dass sein Kredit von vier Ether bei der Auszahlung auch wirklich noch genügend Wert hat. Umgekehrt könnte eine Kreditrückzahlung mit Zins sogar weniger Wert sein, als der zugrunde liegende Kredit.

Aus diesem Grund muss die Kreditvergabe in einer Realwährung geschehen. Damit der Smart Contract feststellen kann, ob die eingezahlte Anzahl Ether auch der vereinbarten Summe entspricht, muss ein Oracle die Währungskurse bekannt geben. Dieses Oracle wird bei der Bezahlung eines Kredites verwendet, um im GUI darzustellen, wie viel Ether gesendet werden muss (Punkt 1 in Abbildung 2). Danach wird bei jeder Einzahlung an den Smart Contract die überwiesene Ether-Summe in die Realwährung konvertiert, um die Korrektheit der Überweisung festzustellen (Punkt 2 respektive 3).

Der Benutzer kann zwischen den Währungen Euro, US Dollar und Schweizer Franken wählen.

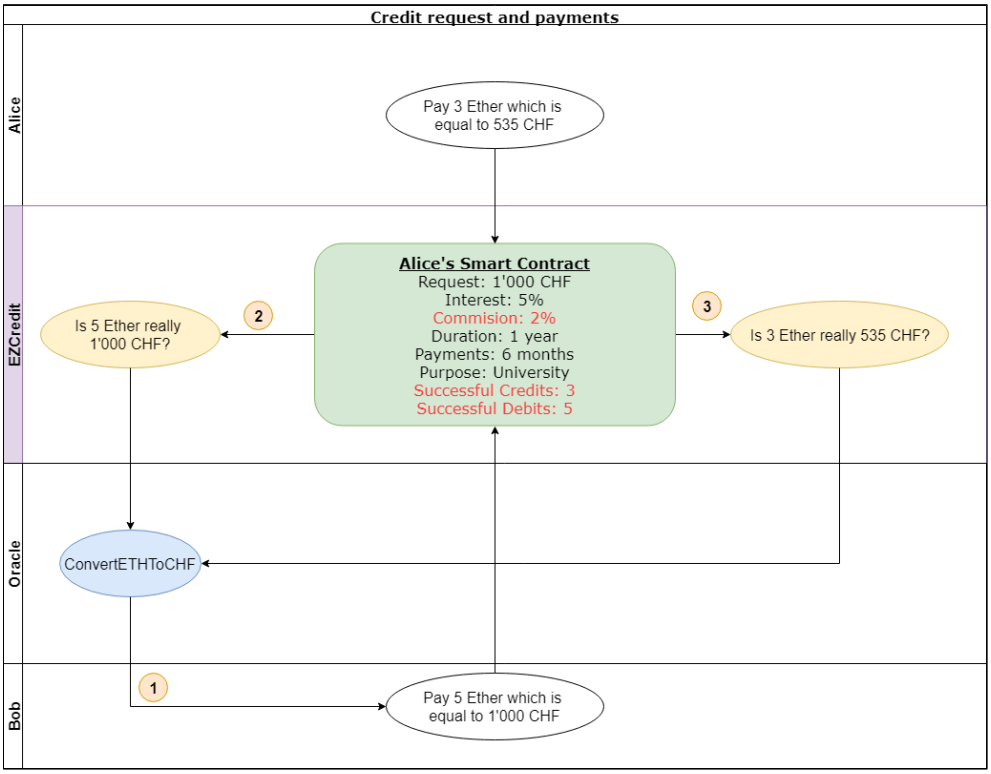

Abbildung 2: Credit Request and Payments. Mit Oracle.

4. AKTIVITÄTSDIAGRAMM

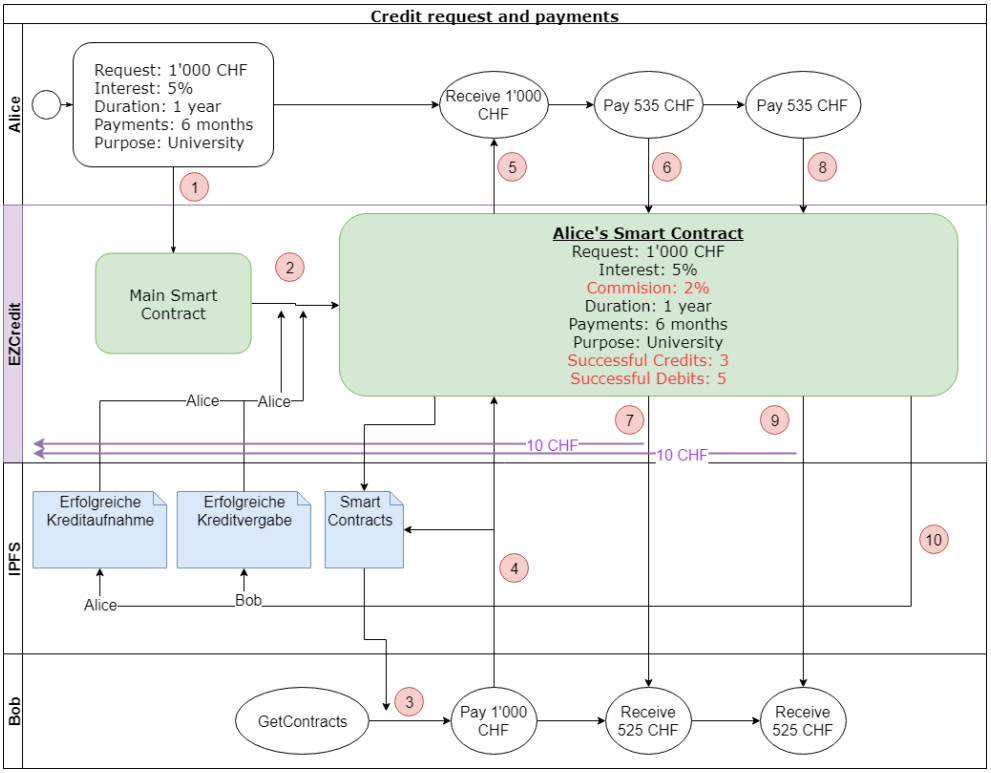

Abbildung 3: Aktivitätsdiagramm

Alice möchte gerne einen Kredit aufnehmen (Siehe Abbildung 3).

(1) Sie gibt ihre Daten über das EZCredit-GUI ein und sendet diese ab. Die Daten werden vom Smart Contract entgegengenommen und aus dem dezentralisierten Speichernetzwerk IPFS werden weitere Angaben zu Alice geholt.
(2) Mit dieser Datenbasis wird dann ein Smart Contract erstellt, welcher zusätzlich zu dem von Alice angegebenen Zinssatz eine Kommissionsgebühr ausweist, welche bei einem erfolgreichen Abschluss für EZCredit abgezweigt wird.
(3) Bob will nun jemandem einen Kredit gewährleisten und fragt nach einer Liste von Smart Contracts.
(4) Er entschliesst sich, Alice den gewünschten Kredit zu gewähren und zahlt den nötigen Betrag an Alices Smart Contract, was auch in der Liste der Smart Contracts vermerkt wird.
(5) Der Smart Contract überweist Bobs Geld an Alice.
(6) Alice zahlt nun im vereinbarten Intervall das Geld zurück, wobei der Zins und die Kommissionsgebühr ebenfalls auf die einzelnen Zahlungen aufgeteilt werden.
(7) Nachdem der volle Betrag abbezahlt wurde, wird für Alice und Bob eine erfolgreiche Kreditaufnahme, respektive eine erfolgreiche Kreditvergabe vermerkt.

Ein alternativer Ablauf ist in Abbildung 4 dargestellt.

(1) Stellt Bob fest, dass ihm nach dem vereinbarten Zahlungsdatum noch kein Geld überwiesen wurde, kann er an den Smart Contract appellieren.
(2) Wenn Alice tatsächlich nicht eingezahlt hat, wird die Adresse von Alice in den Pool der gesperrten Adressen eingefügt und als Folge kann sie keine neuen Kreditanfragen stellen und ihre Contracts werden aus der Liste aller Smart Contracts gefiltert, so dass sie bei einer Abfrage nicht erscheinen. Natürlich wird Bob trotzdem eine erfolgreiche Kreditvergabe angerechnet.

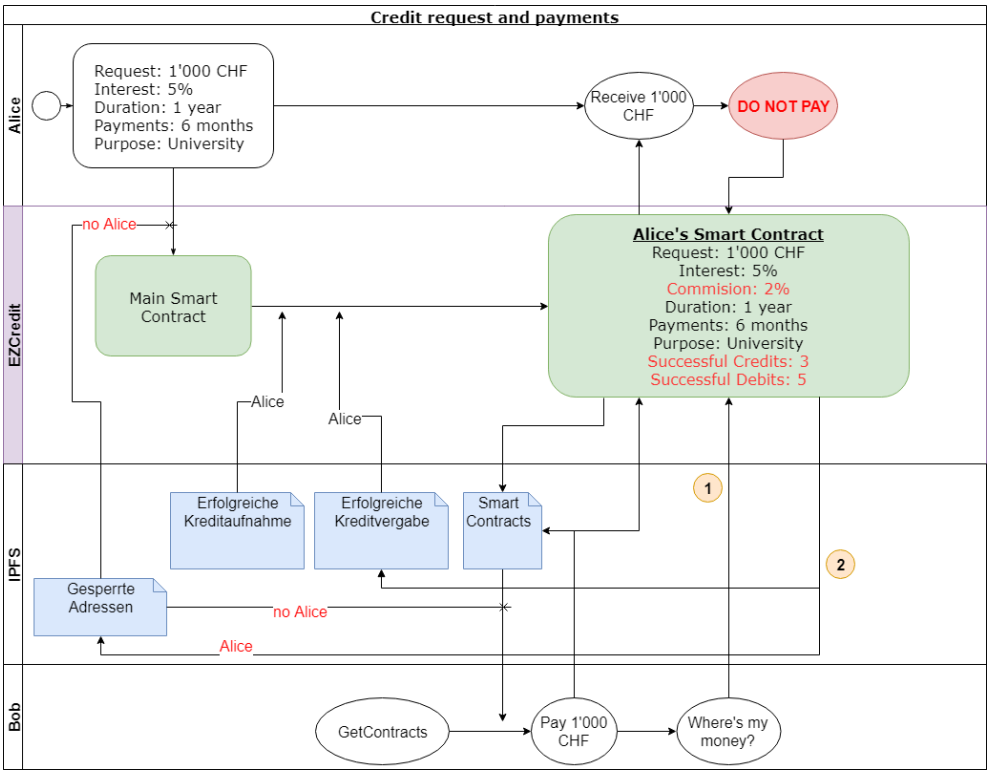

Abbildung 4: Aktivitätsdiagramm. Alternativer Ablauf.

IMPLEMENTATION

Im Folgenden wird die Implementation der DAPP des fiktiven Unternehmens EZCredit vorgestellt. Die umgesetzte Funktionalität, sowie Problemstellungen werden erläutert. Relevante Code Fragmente und Erläuterungen dazu werden aufgezeigt und Themen Sicherheit und Kosten kurz angeschnitten.

1. UMGESETZTE FUNKTIONALITÄT

Die angedachte Lösung ist sehr komplex und braucht für die komplette Umsetzung einiges an Know-How und Zeit. Im Rahmen der Projektarbeit wurde aus zeitlichen Gründen nur der Haupt Smart Contract programmiert. An diesen können Benutzer Kreditanfragen stellen und es wird ein spezifisch für sie angelegter Smart Contract auf der Chain veröffentlicht. Durch diesen Contract kann nun ein interessierter Kreditor den gewünschten Betrag ausleihen. Der Betrag wird dann an den Debitor weitergegeben, welcher nun in einem zuvor festgelegten Intervall Rückzahlungen ausführen muss. Wenn eine Zahlung ausbleibt, kann der Kreditor den Debitor sperren.

2. FEHLENDE FUNKTIONALITÄTEN

2.1 Off-Chain Speicher

Momentan werden alle Daten direkt im Ethereum Netzwerk gespeichert. Bis zum Release muss noch die Integration mit IPFS (Interplanetary File System) implementiert werden, damit Daten wie zum Beispiel die Adressen aller vorhandenen Smart Contracts und die gesperrten Benutzer off-Chain verwaltet werden können.

2.2 Oracles zur Preisbestimmung

Das Ziel der entstehenden DAPP ist, dass der Wert des Kredits und der Rückzahlungen jeweils mit einer FIAT-Währung festgelegt wird. So beantragt der Debitor einen Kredit von 100 CHF und erhält dann Ether im Wert von 100 CHF. Wenn er zurückzahlen soll, muss er nicht den gleichen Ether Betrag, sondern den Wert von 100 CHF zurückzahlen.

Für diesen Zweck muss ein Oracle (siehe Architektur, Kapitel 3) eingesetzt werden, welches den aktuellen Umwandlungskurs von einer Trading-Seite holt. Bei der Implementation dieser Funktionalität traten Hürden auf (siehe Implementation, Kapitel 3) und daher wird in der aktuellen Version ohne Umwandlung gerechnet.

2.3 Speichern und Abrufen der erfolgreichen Transaktionen

Damit ein Kredit sieht, wie vertrauenswürdig ein Debitor ist, muss er seine vergangenen Transaktionen kennen. Dazu werden für jeden Benutzer die Anzahl erfolgreichen Kreditgaben und Kreditrückzahlungen gespeichert.

Dies muss noch umgesetzt werden, wenn die Integration mit dem IPFS gemacht wird.

2.4 Auszahlen der Kommissionsgebühren an Entwickler Account

Zurzeit werden die Kommissionsgebühren an den Kreditor weitergegeben, da noch kein Entwickler Wallet existiert. Vor der Auslieferung der DAPP müssen die Kommissionen jeder Rückzahlung jeweils an einen Entwickler Account weitergeleitet werden.

2.5 GUI zur Interaktion mit den Smart Contracts

Es muss noch ein Web-GUI, respektive eine Smartphone App erstellt werden, die eine Interaktion mit dem Haupt Contract zulässt, damit Kreditanfragen erfasst werden können. Ausserdem soll sie dem Benutzer die Möglichkeit geben, bestehende Kreditanfragen aufzulisten, welche seinen Kriterien entsprechen. Natürlich soll auch eine Verwaltung bestehen, welche alle vom Benutzer erfassten Kreditanfragen und Kreditgaben auflistet.

3. PROBLEME

3.1 Oraclize

Beim Versuch Oraclize zu verwenden, um den Ether Preis über die CoinGecko API abzufragen, trat das Problem auf, dass dieser Call asynchron gemacht werden muss:

```
contract EZCreditPriceTicker is usingOraclize{
    EZCredit private ezCredit;

    function getPrice() public payable{
        ezCredit = EZCredit(msg.sender);

        if (oraclize_getPrice("URL") <= address(this).balance) {
            oraclize_query("URL", "json(https://api.coingecko.com/api/v3/simple/price?ids=ethereum&vs_currencies=chf&include_24hr_vol=false&include_last_updated_at=true).ethereum.chf");
        }
    }

    function __callback(bytes32 myid, string memory result) public{
        assert(msg.sender != oraclize_cbAddress());
        ezCredit.giveCreditCallback(result);
    }
}
```

Abbildung 5: Versuch der Oracle-Implementation

Da aber die aufrufende Funktion den Kurs für die weiteren Berechnungen braucht, war nicht klar, wie der Oraclize Callback nun helfen sollte, die ursprüngliche Transaktion auszuführen oder abzubrechen.

3.2 Web Interface

Aus zeitlichen Gründen war es nicht möglich ein ausgebautes Web Interface zu erstellen, weshalb lediglich die Hauptseite in Code geschrieben wurde. Ein individueller Contract wird auf der Chain generiert und der User kann seine aktuellen Kredit Anfragen sehen. Hier war der Faktor Zeit das Hauptproblem.

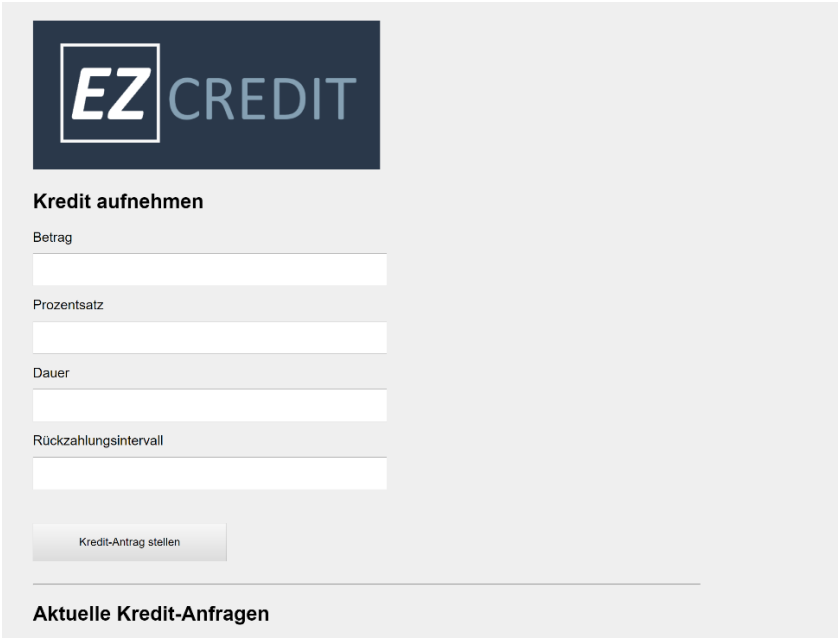

Abbildung 6: Prototyp Web Interface

4. CODE

4.1 Beantragen eines Kredits

Die Funktion in Abbildung 5 kann auf dem Haupt Contract `EZCreditFactory` aufgerufen werden. Es wird sichergestellt, dass sinnvolle Werte mitgegeben werden und der Debitor nicht gesperrt ist. Anschliessend wird mit den zur Verfügung stehenden Werten ein neuer Smart Contract `EZCredit` aufgesetzt. Ein Event wird benutzt, um die Adresse des soeben generierten Contracts auszugeben.

```
function requestCredit( uint requestAmountInFinney, uint8 interestPermille,
                        uint8 durationOfContract, uint8 durationPerPayment) public{
    assert(requestAmountInFinney != 0);
    assert(durationOfContract != 0);
    assert(durationPerPayment != 0);

    // Make sure the debtor is not blocked.
    for(uint i=0;i<blockedAddresses.length;i++){
        assert(msg.sender != blockedAddresses[i]);
    }

    // ToDo: Check the successful credits and debits from the debtor.
    // ToDo: Use oracles to change Ether into FIAT currency.

    EZCredit ezCreditContract = new EZCredit(   msg.sender, requestAmountInFinney, interestPermille,
                                                durationOfContract, durationPerPayment, 20, 0, 0);

    contracts.push(ezCreditContract);

    // Raise an event with the address of the created contract.
    // This event can be read by an outstander and thus the new contract address can be retrieved.
    emit contractCreated(ezCreditContract);
}
```

Abbildung 7: Funktion, welche einen neuen Smart Contract für die Kreditanfrage aufsetzt

Der Konstruktor des erstellten EZCredit-Contracts nimmt die Werte und berechnet einige zusätzliche Informationen:

```
constructor(address payable _debtor, uint _requestAmountInFinney, uint8 _interestPermille, uint8 _durationOfContract,
        uint8 _durationPerPayment, uint8 _commision, uint _successfulCredits, uint _successfulDebits) public{
    assert(_debtor != address(0));
    assert(_requestAmountInFinney != 0);
    assert(_durationOfContract != 0);
    assert(_durationPerPayment != 0);

    debtor = _debtor;
    commision = _commision;
    successfulDebits = _successfulDebits;
    successfulCredits = _successfulCredits;
    requestAmountInFinney = _requestAmountInFinney;
    interestPermille = _interestPermille;
    durationOfContract = _durationOfContract;
    durationPerPayment = _durationPerPayment;

    outstandingDebt = requestAmountInFinney + (requestAmountInFinney * (interestPermille + commision) / 1000);
    amountToPayBack = outstandingDebt;
    amountPerPeriod = amountToPayBack / (durationOfContract / durationPerPayment);

    ezCreditFactory = EZCreditFactory(msg.sender);
}
```

Abbildung 8: Konstruktor des EZCredit-Contracts

4.2 Kredit vergeben

Die Funktion `giveCredit` wird bei einer Kreditvergabe aufgerufen.

```
function giveCredit() public payable {
    assert(creditor == address(0));

    startTime = now;
    creditor = msg.sender;
    debtor.transfer(msg.value);
}
```

Abbildung 9: Bezahlung eines Kredits

4.3 Kredit zurückzahlen

Ein Debitor muss in einem festgelegten Intervall Teilrückzahlungen machen. Der Contract regelt, dass der Debitor nicht mehr als nötig, aber auch nicht weniger als abgemacht zurückzahlen kann.

```
function payback() public payable{
    assert(creditor != address(0));
    assert(outstandingDebt > 0);

    uint neededValue = amountToPayBack / (durationOfContract / durationPerPayment);

    if(neededValue > outstandingDebt){
        neededValue = outstandingDebt;
    }

    assert(msg.value / (1 finney) >= neededValue);

    // Don't let the debtor pay too much.
    if(outstandingDebt >= (msg.value / (1 finney))){
        outstandingDebt = outstandingDebt - (msg.value / (1 finney));
        creditor.transfer(msg.value);
    }
    else{
        uint valueToSend = outstandingDebt * (1 finney);
        outstandingDebt = 0;
        creditor.transfer(valueToSend);
        debtor.transfer(msg.value - valueToSend);
    }
}
```

Abbildung 10: Funktion, um einen Kredit zurückzuzahlen

4.4 Debitor sperren lassen

Wenn der Kreditor bemerkt, dass die abgemachten Rückzahlungen nicht erfolgen, kann er auf dem Contract den Debitor sperren lassen. Es wird dann überprüft, wieviel der Debitor bis zu diesem Zeitpunkt gezahlt hat und wieviel er gezahlt haben sollte. Wenn der Debitor mit seinen Zahlungen in Verzug ist, wird auf dem Haupt-Contract die Sperrung des Debitors eingeleitet.

```
function blockDebtor() public{
    assert(debtor != address(0));
    assert(creditor != address(0));

    uint paymentsDue = (now - startTime) / durationPerPayment;
    uint ownedAmount = paymentsDue * amountPerPeriod;

    // If at this time the debtor has not made enough payments, block him.
    if(amountToPayBack - outstandingDebt < ownedAmount){
        ezCreditFactory.blockDebtor(debtor);
    }
}
```

Abbildung 11: Funktion, um einen nicht zahlenden Debitor sperren zu lassen

Diese Funktion kann nur von bekannten EZCredit Adressen ausgeführt werden:

```
function blockDebtor(address debtor) public{
    for(uint i=0;i<contracts.length;i++){
        // Only a known contract can block someone.
        if(address(contracts[i]) == msg.sender){
            blockedAddresses.push(debtor);
            return;
        }
    }
}
```

Abbildung 12: Haupt Contract Funktion zur Sperrung eines Benutzers

5. SICHERHEITSASPEKT UND KOSTEN

Wie mit Sicherheit garantiert werden kann, dass ein Kreditnehmer diesen auch tatsächlich mit Zinsen an den Kreditgeber zurück zahlt wurde während des Projekts ausführlich diskutiert. Schliesslich kam die Idee auf mit dem Blockverfahren durch einen anderen User, wenn der zurück gezahlte Betrag am Ende der Zahlungsperiode kleiner dem geschuldeten Betrag ist. Durch den Einsatz von Push-Benachrichtungen sollte ein Kreditgeber so Mahnungen an einen Kreditnehmer versenden können.

Das interne Bonitätssystem könnte ausserdem weiter ausgebaut werden, so dass User erst ab einer bestimmten Anzahl von erfolgreich zurückgezahlten Krediten, Beträge über 100 CHF anfragen dürfen. So können sich User schrittweise eine gute oder eben schlechte Reputation aufbauen. Durch die Kommission versucht EZCredit Verluste bei Krediten unter 100 CHF zu versichern.

FAZIT UND LESSONS LEARNED

Einen Smart Contract aufzusetzen ist sehr zeitaufwändig und sollte nicht von Beginnern vorgenommen werden. Da ein in der Blockchain gespeicherter Contract nicht mehr verändert werden kann, müssen intensive Code Reviews und Sicherheitstest von Spezialisten durchgeführt werden, bevor der Contract genutzt werden kann.

Die Erstellung einer Web-Applikation, welche mit der Blockchain interagiert, ist initial ziemlich kompliziert und benötigt tiefgreifendes Verständnis über die Blockchain-Technologie. Da man DAPPS nicht debuggen kann, ist auch der Fehlerfindungs-Prozess aufwändig.

Leider kamen wir nicht mehr dazu, eine dezentrale Speicherlösung zu implementieren und die Verwendung von Oraclize funktionierte nicht wie geplant. Der technische Aspekt allgemein war stellte eine grosse Herausforderung dar.

Insgesamt war es aber ein spannendes Projekt, mit welchem wir unsere ersten Schritte in der Solidity Programmierung unternommen haben.

ABBILDUNGSVERZEICHNIS