Book‘n’Block

Inhaltsverzeichnis

Gegebenes Szenario

Minimum

- 1 Zimmer: kann immer wieder vermietet werden
- 1 Mieter: mietet ein Zimmer
- 1 Vermieter: vermietet ein Zimmer

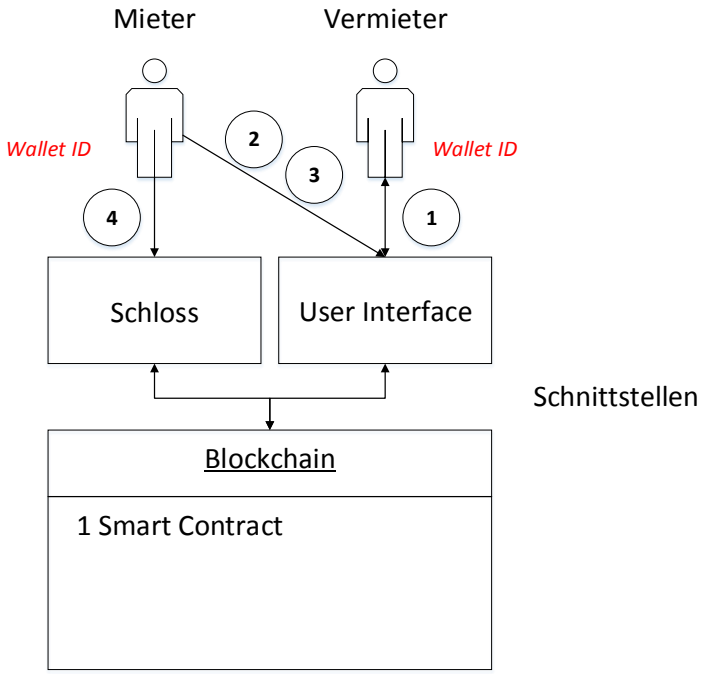
Anmerkungen:

- Ein Zimmer entspricht einer Tür.
- Es ist die Möglichkeit gegeben, während des gebuchten Zeitraums das Schloss mehrmals zu öffnen (Umsetzung mit Whisper).
- Das Schloss soll als Hardware vorhanden sein.

Erweiterung

- Mehrere Zimmer
- Mehrere Mieter
- Mehrere Vermieter

Grafik



1 Vermieter legt Angebot über User Interface in Blockchain an (= Transaktion)
2 Mieter schaut, welche Angebote es gibt (Only Read) und entscheidet sich ggf.
3 Mieter führt Buchung durch (Buchung kann nicht rückgängig gemacht werden und das Geld geht an den Vermieter)
4 Mieter kann Schloss mit Smartphone öffnen

Anmerkungen:

- Es existiert in der Blockchain ein Smart Contract, welcher in einer Datenbank verwaltet wird, um die Daten immer auf den aktuellsten Stand zu halten.
- In einem Smart Contract in der Blockchain wird immer das gleiche Booking durchgeführt.
- Wie sieht die Kommunikation zwischen User Interface und Blockchain aus?
- Ggf. im User Interface auswählen welche Technologie eingesetzt werden soll (Ethereum oder Hyperledger).
- *Spätere Erweiterung von Nummer 3:* Vermieter kann in Angebotserstellung einstellen, wie gut die Bewertung eines Mieters sein muss, damit der Smart Contract die Buchung akzeptiert.
- Jede Tür / jeder Benutzer verfügt über eigenen Node, welcher auf einem Server bzw. Handy läuft.

Mietzimmer

Angebot

Variable	**Datentyp**
Angebot ID	Long
Tür ID	Long
Mietpreis (pro Nacht)	Long
Gültigkeitszeitraum Inserat (Zeit von / Zeit bis)	Date (Verwendung von UNIX)
Adresse des Objekts	Text
Name des Vermieters	String
Vermieter Wallet ID → *Public Key*	ca. 20 zufällige Zeichen
Beschreibung	String
evtl. Zimmerbild	

Buchung

Variable	**Datentyp**
Angebot ID	Long
Buchungszeitraum Zimmer (Check-in / Check-out)	Date (Verwendung von UNIX)
Mieter Wallet ID → *Public Key*	ca. 20 zufällige Zeichen

Anmerkungen:

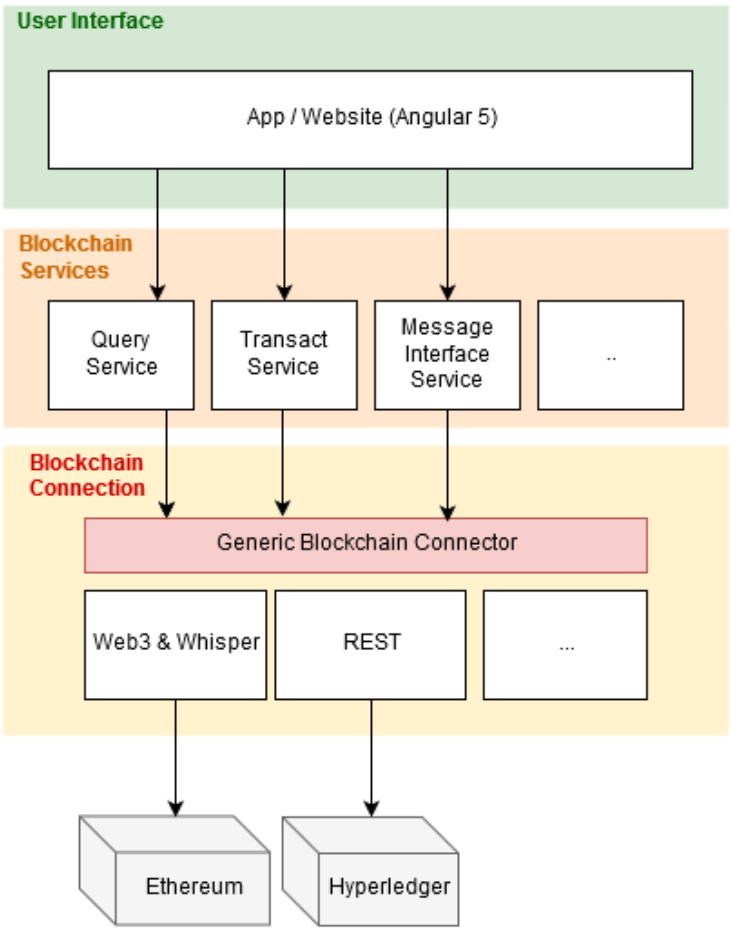
- Tür ID ist Zieladresse für Whisper (quasi auch *Public Key*).
- Gültigkeitszeitraum Inserat wird durch den Vermieter festgelegt (Zeitraum kann auch leer gelassen werden und ist dadurch bis zur Löschung gültig).
- Buchungszeitraum Zimmer wird durch den Mieter festgelegt.
- Mieter Wallet ID und Vermieter Wallet ID sind „quasi“ die Benutzernamen.
- *Public* und *Private Key* werden automatisch generiert und werden für die Transaktionen benötigt.
- Mieter Wallet ID und Vermieter Wallet ID in JSON-File oder Datenbank (Serverseitig) speichern?

Architektur

Architektur der Anwendung

Im Folgenden wird die Architektur der BookNBlock Anwendung näher beschrieben. BookNBlock ist eine Serverless App. Das bedeutet, der komplette Datenfluss und die Interaktion geschieht ausschließlich zwischen dem Client und der Blockchain.

Client Architektur



Zur besseren Übersicht und um verschiedene Bereiche der Anwendung parallel entwickeln zu können, wird der Client in drei verschiedene Layer unterteilt.

Userinterface Layer

Eine Angular 5 Applikation bietet die Möglichkeiten Zimmer zur Vermietung anzubieten und ein Zimmer zu buchen. Die Applikation ist der zentrale Marktplatz für die Vermietung und das Mieten von Zimmern.

Blockchain Services Layer

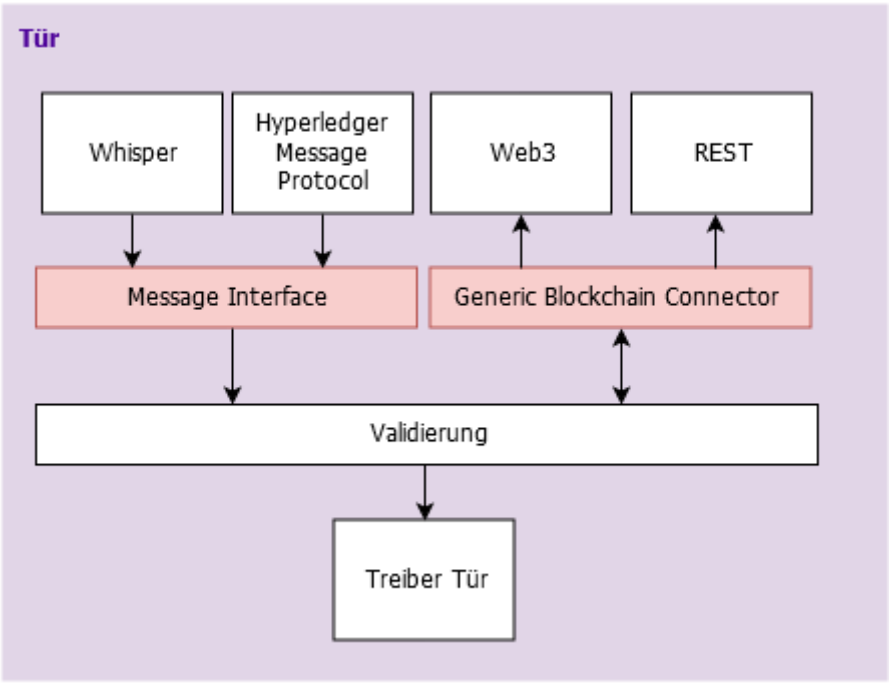
Aus Gründen der Skalierbarkeit und Ausfallsicherheit werden verschiedene Blockchain Services verwendet, um aus dem User Interface mit der Blockchain zu interagieren. Jeder Service befasst sich mit einer definierten Fachlichkeit. Weitere Services sind denkbar. Zum jetztigen Zeitpunkt sind die folgenden Services geplant:

- *Query Service* ist verantwortlich um Informationen von der Blockchain abzufragen
- *Transact Service* ist verantwortlich für das Erstellen von Transaktionen in die Blockchain
- *Message Interface Service* ist verantwortlich für das Senden von OpenDoorMessages über das Ethereum Netzwerk um die Tür eines Zimmers zu öffnen

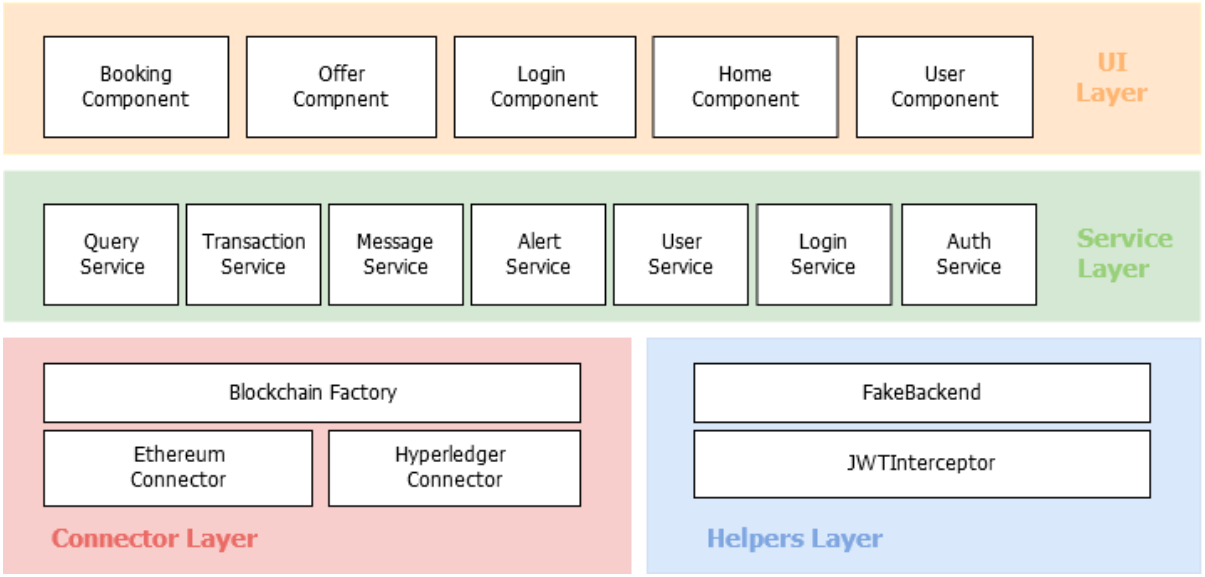
Blockchain Connection Layer

Um die tatsächlichen Zugriffe auf die Blockchain zu kapslen, wird ein Generic Blockchain Connector verwendet. Dieser bietet eine einheitliche Schnittstelle, da unsere Anwendung mit zwei verschiedenen Blockchain Technologien (Ethereum und Hyperledger) verwendet werden kann. Die Interaktion mit der Ethereum Blockchain wird mittels Web3 realisiert. Der Zugriff auf die Hyperledger Blockchain mittels REST. Zudem wird ein Message Mechanismus für jede Blockchain Technologie bereitgestellt. Im Falle Ethereum wird Whisper verwendet, für Hyperledger ist zu diesem Zeitpunkt noch nicht genau klar, welches Protokoll verwendet wird.

Architektur Tür

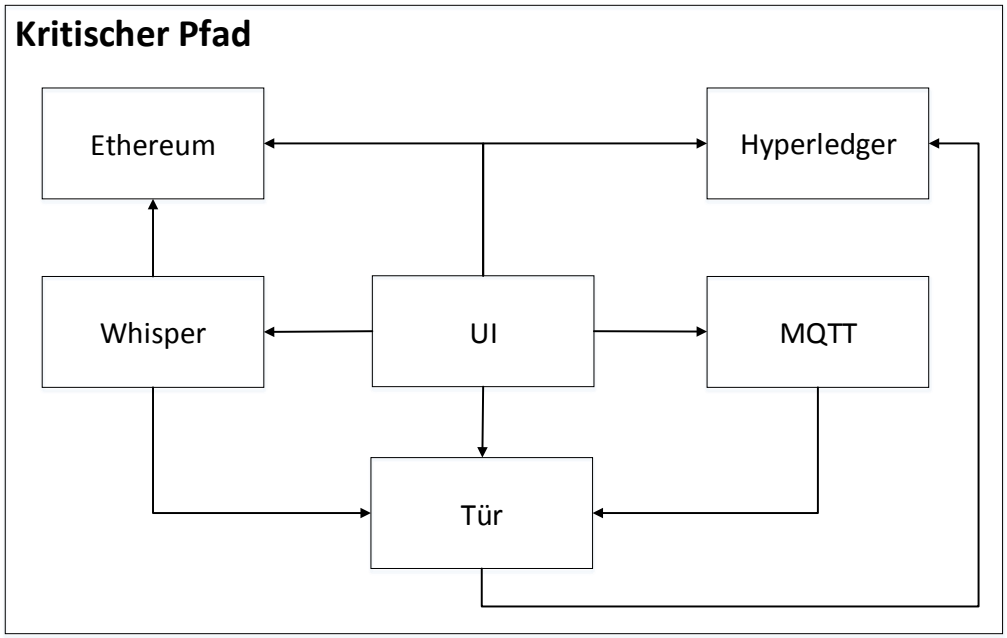


Architektur UI



Umsetzung

Modell des Kritischen Pfads

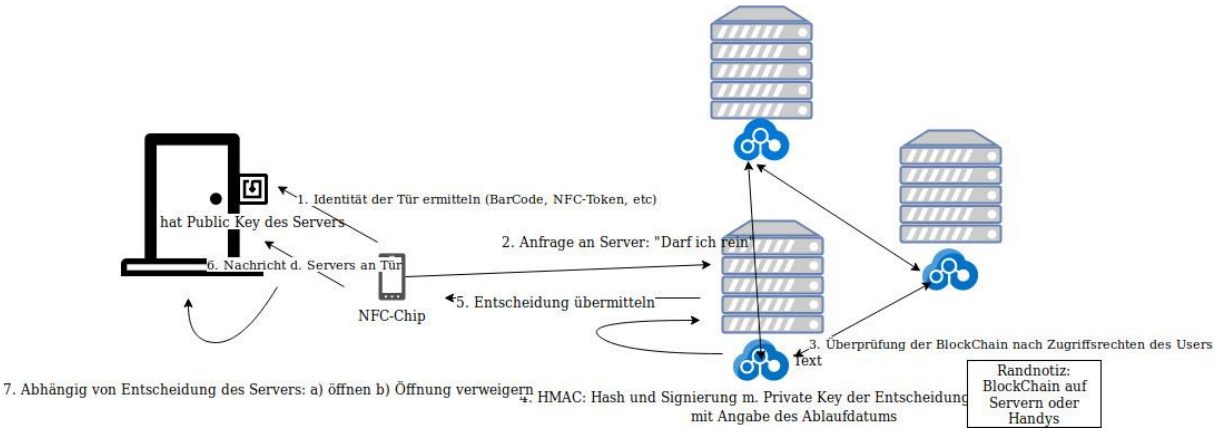


Smart Contract mit Ethereum

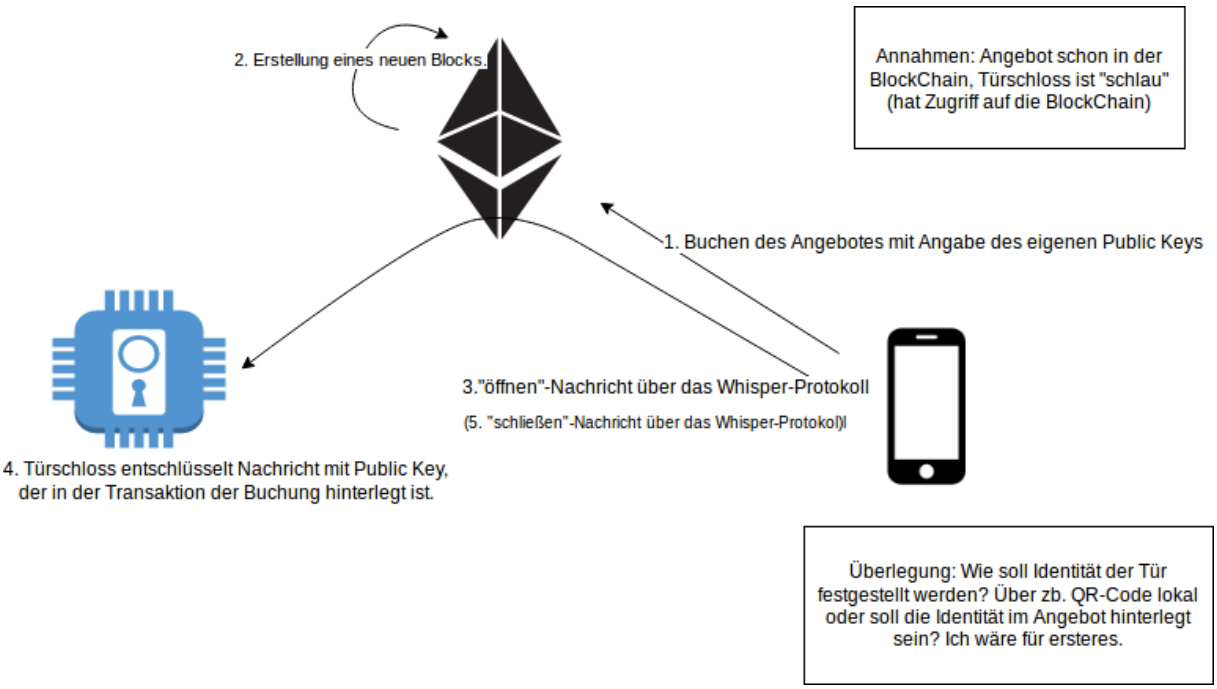
- Ein oder mehrere Smart Contracts pro Angebot
- Verwaltung von Public Keys (Ethereum-Seite) und überlegen wie diese in Block geschrieben werden
- Umsetzung mit dem Framework Truffle + Ganache
- Smart Contract ID referenziert immer auf Ursprungsblock
- Welche Aktionen werden innerhalb des Smart Contracts ausgeführt?

Whisper Messages Protokoll für Ethereum

Dumb Door



Smart Door



- Keine Transaktionen (und dadurch keine Kosten)
- Integriertes Schloss (Hardware) → Zugriff über Blockchain
- Smartphone sendet Token, welches zur Authentifizierung des Mieters durch den Vermieter dient
- Whisper signiert und hinterlegt einen PrivateKey

web3.js Client bei Ethereum Blockchain

- Umsetzung mit Angular und einem Node
- Anbindung der Daten in der Blockchain allgemein
- Kommunikation zwischen Blockchain und User Interface

REST-Schnittstelle bei Hyperledger Blockchain

- Anbindung der Daten in der Blockchain allgemein
- Kommunikation zwischen Blockchain und User Interface

User Account für UI

- Eigene Verwaltung von Accounts, Geld und Keys
- Plugin oder direkt über Ethereum bzw. Hyperledger Wallet

User Interface

- Schickt Nachrichten an die Tür
- Webinterface (grafische Benutzeroberfläche)
- Frontend Darstellung / wie buche ich überhaupt?

Hyperledger

Siehe https://github.com/darenegade/BookNBlock/blob/master/Hyperledger/README.md

Wichtig ist, dass Anpassungen für die Türe und das Frontend durchgeführt werden müssen.

In unserem Projekt streben wir an, Hyperledger über einen Server zu Nutzen. Hierzu muss die REST-Schnittstelle genutzt werden, um mit der Blockchain zu kommunizieren. Dazu müssen die Befehle im Skript unter der Funktion getRestService() manuell ausgeführt werden.

Aktuelle Features:

Bezeichnung	**Funktion**
Angebot anlegen	insertOffer
Angebot löschen	delete
Angebot über seine ID finden	getOffer
Angebote über Public Key suchen	queryOffersByPk
Anzeigen der Historie	getHistoryForOffer

Geplante Features:

- Angebot übertragen
- Angebot mieten
- Optimierung
- ABAC (Attribute-Based-Access-Control)
- Kafka / ZooKeeper

Transaktion durchführen

- Hyperledger: Private Key, Username (Wallet ID? Fingerprint) / E-Mail Adresse
- Ethereum: Key Phrase → Private Key, Wallet IDs / E-Mail Adresse
  - Schnittstelle zum Handy gibt es fertig als App

Schnittstellen des Smart Contracts

Angebot einstellen / löschen / mieten:

- `insertOffer(uint price, string ownerName, .....)`
  - Angebot mit sämtlichen Parametern eines Angebotes aus dem Datenmodell (siehe Angebots-Tabelle auf Seite 4) einstellen
- `deleteOffer(uint offerID)`
  - Angebot mit bestimmter ID löschen
- `rentAnOffer(uint checkIn, uint checkOut, .....)`
  - Zimmer mit sämtlichen Parametern einer Buchung aus dem Datenmodell (siehe Buchungs-Tabelle auf Seite 4) mieten

Iterator für Buchungen, da diese nur von den betroffenen Ver-/Mietern eingesehen werden sollen:

- `getOfferIDs() return uint[]`
  - IDs aller Angebote erhalten
- `getFreeOfferIDs(uint256 from, uint256 to) return uint[]`
  - IDs der freien Angebote erhalten
- `getOffer(uint offerID) return (uint price, string ownerName, .....)`
  - genaue Informationen zu einem Angebot erhalten
- `getBookingIDsForOffer(uint offerID) return uint`
  - zu einem Angebot alle Buchungen anzeigen
- `getBooking(uint bookingID) return (uint checkIn, uint checkOut, .....)`
  - Daten zu einem gebuchten Angebot erhalten (siehe Buchungs-Tabelle auf Seite 4)

Tür muss wissen, ob der Mieter die Erlaubnis hat die Tür mit einer bestimmten ID zu öffnen:

- `isAllowedAt(uint bookingID, address renterID, uint256 time) return bool`
  - Zugriff von Mieter auf Tür prüfen

Authentifizierungskonzept

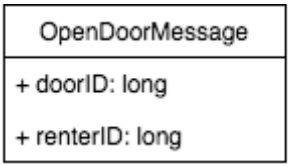
- Nicht mit Ethereum Log, sondern extra Login zum Anmelden
- Problem: Eingabe PrivateKey
- Stand 29.05.: Eingabe Ethereum-Adresse und PrivateKey
- Benötigt wird ein eigener Contract für das Mappen einer Login-Adresse auf eine Ethereum-Adresse.
  - wenn dann nur Metamask
  - geht nicht für Hyperledger

- Stand 21.06.: Registrierung mit E-Mail und Passwort
- Nutzerprofil enthält Private Key, Passphrase und Abfrage, ob Ethereum oder Hyperledger verwendet werden möchte (letzteres ist momentan optional: nur grundsätzliche Umsetzung)

Einsatz von Rinkeby

Rinkeby ist ein sogenanntes „testnet“, dass die Möglichkeit bietet, kostenlos Smart Contract Lösungen einzubinden und zu testen. Es ist sozusagen eine gebührenfreie Kopie des Ethereum-Netzwerks. Nach der Registrierung bei Rinkeby ist es möglich, Ether zu schöpfen (täglich rund 20 Ether) und anschließend Transaktionen durchzuführen. Für unsere Smart Contract Lösung wird Rinkeby für die Mieter und Vermieter Wallets und deren Kommunikation miteinander eingesetzt.

Open Door Message



- Grundstruktur der Nachricht:

```
{
 "doorID": "Die ID der Tür",
 "timestamp": "UNIX in ms",
 "renterPubkey": "PublicKey des Mieters (verschickt die Nachricht)"
}
```

- Protokoll Ethereum: *Whisper*
- Protokoll Hyperledger:
  - Installierter *MQTT*Broker in der GoogleCloud (erreichbar über `104.196.103.14:1883`)
  - Linux Systeme: Client zum Testen und Debuggen installieren (Befehl: `apt-get install mosquitto-clients`)
  - Subscribe: `mosquitto_sub -h 104.196.103.14 -t test`
  - Publish: `mosquitto_pub -h 104.196.103.14 -t test -m "Hello World"`
  - Nachricht:

    ```
    Adresse: URL des MQTT Brokers
    Topic: 'door'
    ```

  - Zusätzlich muss die Nachricht beim MQTT Protokoll teilweise signiert werden.

    ```
    {
     "doorID": "Die ID der Tür", # entspricht der offerID und diese
      ist eindeutig
     "timestamp": "UNIX in ms", | dieser Teil ist Signiert mit dem
      PrivateKey des Mieters
     "renterPubkey": "Der PublicKey des Mieters (der die Nachricht
      verschickt)", | die Werte werden hier mit ',' separiert
    }
    ```

  - Die Signierte Nachricht schaut wie folgend aus:

    ```
    {
     "doorID": "Die ID der Tür",
     "payload": Signierter Hash mit 'timestamp' und 'renterID' und
      Komma separiert
    }
    ```

Offene Punkte

- **Whisper:** Datentypen und Interfaces abgleichen
- **Integration Tür:** Aktuell nur auf Konsole laufend / Alternativ Skript zum Türöffnen
- **UI:** Integration UI → Whisper: Compile
- **UI:** Button für Tür öffnen mit Validierung (nur am PC möglich)
- **UI:** Tür Deployment
- **Hyperledger:** Connector ist in Arbeit
- **Hyperledger:** Türzugriff über Server (Zugriff via VPN etc.)

Anmerkungen:

- **Buildmanagement**: Rene und James?
- Public und Private Key Verwaltung am PC (speichern im localStorage des Browsers)
- 2 Devices für die Zugänge müssen vorhanden sein
  - PC
  - Handy

Meilensteine

Nach Gruppen

Verzögerung: Spätestens bis Wochenende 29.06.-01.07.2018 fertig stellen!!!

Gruppe	Meilensteine	Zuständig	Deadline
Hyperledger	*Schnittstelle Tür* an Max übergeben *Schnittstelle Smart Contract* Funktionsaufrufe *Schnittstelle Web3 Client* Authentifizierung mit Frontend	Deniz Frank	Abhängig von der Verfügbarkeit des Servers, spätestens Wochenende 15.06.-17.06.2018
Tür mit Whisper	*Hardware* *Tür geht auf durch Whisper* *Mit Validierung Contract auslesen* *Kommunikation mit Smart Contract / Web3 integrieren* *Optional: Hyperledger Umsetzung*	Felix Max Peter	Wochenende 15.06.-17.06.2018
Smart Contract	*Contract* *Implementierung mit der Tür* *Implementierung mit dem Frontend* *Einrichten Mieter & Vermieter Wallets mit Rinkeby* *Authentifizierung mit Wallet* Zugriff Blockchain und Wallet *Optional: Contract Test Coverage überprüfen auf Vollständigkeit*	James Rene	Wochenende 15.06.-17.06.2018
Web3 Client	*Kommunikation mit der Tür via MQTT* *Einzelne Zugriffe auf die Hyperledger Blockchain* Angebot einstellen, buchen, anzeigen; User registrieren *UI Anwendung Dialoge* *Login* mit Out O oder selbst implementiert *Kommunikation mit Ethereum*	Alex Daniel Michi	Wochenende 15.06.-17.06.2018

Benötigt wird ein Server (mit Ubuntu 16.04), um Zugang zur Hyperledger Umsetzung zu erlangen, da diese Anwendung nicht unter Windows läuft. Auf diesen Server sollten alle Gruppenmitglieder Zugriff haben, um die Zugriffe dann über eine REST-Schittstelle testen zu können. Momentan läuft auf dem Hochschul-Server ein CentOS.

Nach Umfang der Umsetzung

Tests der Umsetzung	Inhalt	Zuständig	Deadline
Minimum	1 Zimmer 1 Mieter 1 Vermieter	alle	Wochenende 29.06.-01.07.2018
Erweiterung	Mehrere Zimmer Mehrere Mieter Mehrere Vermieter	alle	Wochenende 29.06.-01.07.2018

Organisatorisches

- Pflege von Einzelaufgaben und Einzelnen Projektteilen übernimmt jede Kleingruppe selbst.
- Besprechung immer Dienstag 30-60 Minuten. Danach Arbeiten in Kleingruppen.

Kleingruppen

- Hyperledger: Deniz, Frank
- Tür mit Whisper: Felix, Max, Peter
- Smart Contract: Anna, James, Rene
- Web3 Client: Alex, Daniel, Michi

Veranstaltungstermine

- ~~Dienstag 29.05.2018~~
- ~~Dienstag 05.06.2018~~
- ~~Dienstag 12.06.2018~~
- ~~Dienstag 19.06.2018~~
- Dienstag 26.06.2018

Kurzvorträge

Team-Mitglied	Thema	Datum
Anna	Projektmanagement von „BookNBlock“ – Überblick	26.06.2018
Alex	User Login: Authentifizierungskonzept	29.05.2018
Daniel	OpenChain	24.04.2018
Deniz	Hyperledger	08.05.2018
Felix	Tür Kommunikation mit Smart Contract	12.06.2018
Frank	Hyperledger	08.05.2018
James	Solidity und Testing mit Ethereum	29.05.2018
Max	MQTT	05.06.2018
Michi	Verschlüsselung und Datenschutz Blockchain	26.06.2018
Peter	Whisper + evtl. Tür zeigen	26.06.2018
Rene	Contract Solidity	05.06.2018

Seminararbeit

- Umfang maximal 5 Seiten pro Team-Mitglied (Text und Bilder)
- Kapitel mit Namen kennzeichnen (Namen vorm Fließtext)
- Abgabe (durch Anna) am Freitag den **13.07.2018** per E-Mail als PDF-Datei
- Spätestens bis Montag den **09.07.2018** in das Sharelatex-Template einfügen
  (Anna prüft dann nochmal Rechtschreibung, Layout etc.)

Abschlusspräsentation + Demo

- Unsere Gruppe: Donnerstag **05.07.2018 ab 17 Uhr** (*Pflichtveranstaltung*)
- Andere Gruppe: Dienstag 03.07.2018 ab 17 Uhr (*freiwillige Teilnahme*)
- Vortrag im Team; Dauer ca. 10 Minuten pro Team-Mitglied
- Umfang: Aspekte der Umsetzung/Vergleiche/andere Möglichkeiten/Probleme etc.
- Spätestens bis Sonntag den **01.07.2018** in Google Slides einbauen
  (Anna prüft dann nochmal Rechtschreibung, Layout etc.)
    - schlichte weiße Folien
    - Text: *Calibri* Größe 18 -28

Testen aller Komponenten

Alle zusammen am Montag den **02.07.2018** *oder* Dienstag den **03.07.2018**