Blockchain based Smart Renting

For the university course “Project Open Distributed Systems” on the Technical University Berlin

David Munkacsi
Computer Science (Master)
Budapest, Hungary
david.munkacsi@campus.tu-berlin.de

Abstract

Blockchain is one of the most lucrative buzzword nowadays in the IT branch, its importance cannot be neglected. For many business cases a Blockchain-based solution can come with a lot of advantages. The network provided by Ethereum made their Blockchain network programmable, which extends its usability even more and brings a new interesting approach for application developers.

Developers can create smart contracts to implement the required business logic and deploy them on the Blockchain. This combined with Frontend technologies it is possible to create decentralized applications (DApps), which can provide solution for many business cases.

Predictions show that these technologies will be dominating in many business areas like finance or real-estate. The relevance of Blockchain technologies in real-estate is obvious, the related business processes can be enhanced via the features provided by the Blockchain, like decentralization, almost instantaneous payment or immutability.

The goal of this paper is the detailed documentation of a decentralized application, which implements various use-cases from the real-estate branch with the help of a self-built private Blockchain network, smart contracts and Frontend technologies.

Keywords – Blockchain, Ethereum, Smart contracts, Angular, Decentralized applications, Docker, Solidity

I. INTRODUCTION

Blockchain technologies are becoming more important by day, the interest and curiosity towards them increased extremely over the past few years. This increased interest is based on the multiple advantages, that the Blockchain networks bring with themselves, like decentralized data storage, immutability, almost instantaneous payments.
The expression Blockchain is used inappropriately in a lot of cases. People are thinking that Blockchain is a technology, that can be used to create decentralized applications (DApps) for the different scenarios. Wrong. Blockchain is a distributed storage paradigm, that combines already existing technologies (hashing, distributed networks, consensus algorithms) to store data in a continuously expandable list of blocks consisting of transactions, which is infeasible to falsify, hence the used cryptographic methods.
Numerous systems like Bitcoin or Ethereum have been built, adapting this paradigm, to solve different application scenarios.
Ethereum is the most relevant system for application developers. It makes it possible to deploy program code on the network in form of smart contracts. With the help of smart contracts applications can be developed on the top of a Blockchain for many business purposes. The advantages brought by using Blockchain technologies can be exposed on many areas like finance or real-estate. Real-estate can use them to provide almost instantaneous payments, immutable data or decentralization (banking would not be necessary).
The goal of this paper is to introduce and give a detailed description of a decentralized application, which implements various use-cases from the real-estate branch. The application uses a private self-implemented Blockchain network (private Ethereum network) and realizes the different business scenarios with the help of smart contracts written in Solidity. The Frontend of the application is made with the help of Angular to provide a user interface and to be able to use the implemented functionality.
The structure of this paper is described in the followings. First the used technologies will be introduced briefly. Section 3-4 describes the problem statement and the implemented application use-cases. After that the main components of the applications will be described, with a manual how to start the application on a local machine. The section after that describes the client-side architecture, which is specially designed for smart contracts. The section Network Infrastructure describes the virtualized Ethereum network and roles of the different nodes, which the network consists of. In the Evaluation section the implemented solution, the used technologies and their limitations will be evaluated. The last section Conclusion sums up the experience and remarks of the developer.

II. Used Technologies

A. *Ethereum/Solidity*

Ethereum is a distributed system, which offers the creation, administration and execution of smart contracts in its own Blockchain. It therefore represents an alternative to the classic client-server architecture.
Solidity is an object-oriented, high-level programming language with a JavaScript-like syntax for developing smart contracts for the Ethereum Blockchain platform.
The business logic of the application is implemented with the combination of these two technologies. The smart contracts of the application are written in Solidity and they are deployed to the created private Ethereum network to realize the Backend of the application.

B. *Angular*

Angular is a TypeScript-based front-end web application framework. It is developed by a community of individuals and companies, led by Google, and published as open source software.
In the current application Angular is used to connect to the Ethereum network and invoke the implemented business processes.

C. *Docker, Geth*

Docker is a state-of-the-art platform for isolating applications by virtualizing containers.
Docker makes it easy to deploy applications or infrastructure because containers that contain all the necessary packages are easy to transport and install as files.
Docker is used to deploy and maintain a private Ethereum network, which is based on the *Go Ethereum (Geth)* client, that implements an Ethereum node and uses the protocols, that are required for building a private network. The communication of the nodes and the network infrastructure is implemented in the docker files.

III. Problem Statement

The implemented application is a real-estate portal, which connects landlords and tenants. The landlords can advertise their apartments and the tenants are able to rent these apartments. Why is it a good decision to use the Ethereum network for this business case?
The biggest advantage is the fact, that the whole renting process can be decentralized. Rent usually requires a third centralized entity (bank, real-estate), this can be completely left out, which makes the whole renting process faster and more efficient. The payments over the network are almost instantaneous and safe by design.
Another big advantage is, that the data stored on the Blockchain is immutable. This means that every transaction made can be traced back, which makes it harder to commit fraud. The portal can be extended with IoT use-cases as well. One example would be to connect a smart lock to the apartment, which only lets the tenant to enter the apartment, if he/she paid the rent already.

IV. Implemented Use-Cases

The following use-cases are implemented in the application:

- Basic user management. The user can register and login via the application.
- The user can create and advertise an apartment.
- The user can browse between apartments.
- The user can rent an apartment. The renting process consists of two steps: transferring the deposit and the rent. When the transfers were successful the user will be the temporary owner of the apartment (tenant). If the renting process is cancelled the user will receive a full refund.
- The user can pay his/her rent over the detailed view the apartment.
- The user can browse his/her own apartments or the apartments he/she currently rents.
- The user can terminate his/her contract over the detailed view of the apartment. If the process was successful, the user receives the previously paid deposit back.
- Users can message each other via a chat interface.

Every feature except messaging is implemented via smart contracts. The first approach was to implement the messaging with smart contracts as well. However, the private network, somehow did not allow to connect to it via the web socket protocol. As a fallback plan the architecture of the application was extended with an Express webserver, and the required functionality was implemented through this, with the usage of web sockets.

V. Components

The application consists of four major components:

- Private Ethereum network virtualized via Docker containers.

- Smart contracts that contain the business logic of the application and can be deployed to the private network.
- Express webserver that extends the business logic in some cases (Web sockets, messaging).
- Angular Frontend that connects to the private Blockchain network and invokes the implemented functionality.

Prerequisites to run the application locally:

- NPM and Node must be installed.
- Truffle npm module must be installed. The easiest way is to install it globally via npm module system: *npm install -g truffle*.
- Docker must be installed and started on the local system.

The application can be built and started by executing the following steps:

1. The Docker containers must be built and started. To do this navigate to the *network* directory of the application. With the command "*docker-compose up*" the docker containers will be built and the network will be started. The mining process on the network starts automatically.
2. The next step is to deploy the smart contracts to the network. Navigate to the *ethereum* directory and execute the "*truffle compile"* command to compile the smart contracts. To deploy them to the network, execute the "*truffle migrate"* command. This will execute the implemented migration script and deploy the smart contracts to the target network, which is specified.
3. Navigate to the *server* directory and execute the script inside via the "*node server.js*" command.
4. Execute the "*npm install*" command in the root directory of the application to install the required node modules.
5. Execute the "*npm start*" command to start the Angular application and copy the compiled smart contracts to the Frontend. This is required for the registration of the ABIs (application binary interfaces) and the addresses of the smart contracts.

VI. Client-Side Architecture

The client-side architecture defines the integrability and ease-of-use of the smart contracts on the Frontend. During the design phase of the architecture the goal was to find a way to integrate smart contracts efficiently in the client-side code. The *Web3 API* is the most used client-side library to connect to an Ethereum network (either the main net or a private network) and invoke smart contract functions. However, the API is still rudimentary, so to use it efficiently the architecture around it must be designed properly.
The approach in the application consists of two layers:

- Provider layer – abstract connection to a specific Ethereum network using the Web3 API.
- Contract layer – abstraction of the smart contracts ABIs (application binary interfaces). The contained functions are either calling the implemented smart contract functions or sending transactions to the network.

The classes of both layers must be injectable via Dependency Injection. Every contract has a provider instance, which is an abstract connection to the network, and it is required to send transactions.
The implemented architecture is scalable, connections and contract can be easily injected and used anywhere on the Frontend. With the abstraction of the connection layer it is possible to connect to multiple Ethereum networks. It results in flexibility as well, because smart contracts from another network can be used, without any special knowledge of the network. The injectability makes loose coupling between contracts possible too. The architecture is quite transparent, one contract component is abstracting only one smart contract, which means that the separation of concerns is enforced.

VII. Network Infrastructure

During the design of the private network it was quite important to provide a scalable infrastructure, which can be easily deployed on different machines. The best choice for this demand is the Docker technology, which is perfectly capable to virtualize such a network via different containers. *Go Ethereum* (Geth) is an Ethereum node implementation, which can be containerized and connected to each other to produce a virtualized network.
The nodes in the designed network have the following roles:

- Bootnode – The responsibility of a bootnode to bootstrap other nodes to the peer-to-peer network. Bootnodes usually have static IP addresses, so they are serving as the core of the network.

- Development node – The development node provides an interface to connect to the network either via HTTP or Web sockets. Smart contract functions can be called via the development node, so this is the most relevant role for the application developers.
- Miner node – The miner nodes are solving a difficult cryptographic riddle to extend the chain of blocks. They are providing new coins in the circulation via the mining process as well.
- Explorer – The explorer node is a simple web application over an HTTP interface, which provides information about the Blockchain network like current block mined or account information.

The containers are defined in the Docker configuration with only one command it can be deployed to the target machine.

VIII. EVALUATION

Using a decentralized application is quite suitable for the chosen real-estate application scenario. The advantages, like immutability, safe payments, decentralization, are some considerable features. There are still some major drawbacks, that must be considered, before choosing this path.

The developer community for smart contracts is still rudimentary, it takes a lot of research to solve non-trivial questions during the implementation. The existing APIs are quite low-level, which means, that in some cases it is possible, that the required functionality not even provided so far. In this case a feature or pull request must be made, which can take a lot of time to evaluate and to get into the production code. Overall, the effort of the development is higher than by conventional applications.

The learning effort to understand decentralized applications is vast. It requires a completely different perspective, that slows down the whole development process as well.

Another important lesson learned during development is, that the Blockchain network is not supposed to store massive data sets. The smart contracts cannot return complex collections; hence it is quite hard to design APIs, that requires a list as a result for instance. It is possible to solve this matter by using IPFS (InterPlanetary File System) and store only hash values on the Blockchain. However, it only makes the application architecture more difficult and adds another component to the already complex system.

IX. CONCLUSION

Blockchain-based decentralized applications can provide solution to various business cases. However, it is always required to evaluate first, whether the advantages brought by the technology can be exposed or not. Smart contract implementation is quite recent in the IT world, it is not well documented, the existing APIs are still low-level, so it takes more effort to develop decentralized applications.

For the chosen real-estate business-case the power of Blockchains could be exposed, but the effort that comes with the development must be evaluated, whether it worth to be used or not.

REFERENCES

[1] https://de.wikipedia.org/wiki/Smart_Contract

[2] https://de.wikipedia.org/wiki/Blockchain

[3] https://de.wikipedia.org/wiki/Angular

[4] https://de.wikipedia.org/wiki/Docker_(Software)

[5] https://de.wikipedia.org/wiki/Ethereum

[6] https://de.wikipedia.org/wiki/Solidity

[7] https://web3js.readthedocs.io/en/1.0