

University of Ljubljana
Faculty of Computer and
Information Science



Fog Computing for Smart Services

Assist. Sandi Gec
Assist. dr. Petar Kochovski

Introduction to Ethereum Smart
Contracts

2022
-
2023

28.2.2023



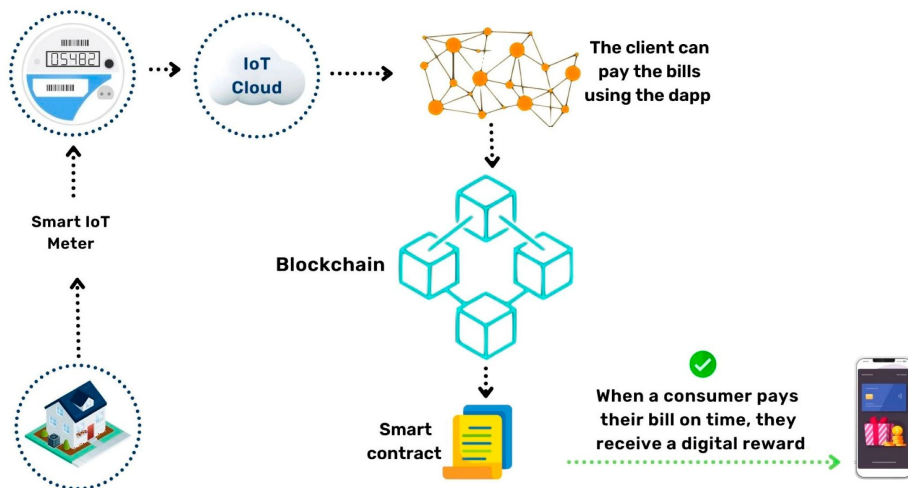
Outline

- **Introduction to decentralized applications (dApps)**
- **Ethereum ledger**
- **Smart contract development in Solidity:**
 - **Structure of the contract**
 - **Base tools**
 - **Development plan**
- **Smart contract examples**
- **Future content**
 - **Setting up a local environment with TruffleSuite and Ganache**
 - **Advanced features:**
 - **Off-chain data interaction**
 - **Migration among different chains**
 - **web3 integration in native HTML and Angular framework**

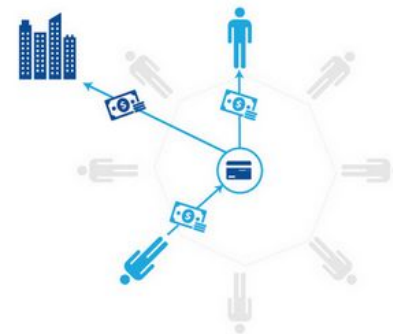


Introduction to dApps ⁽¹⁾

- **What is a decentralized application (DApp)?**
 - The majority of Web applications are **centralized** from the end-user point-of-view.
 - Centralized applications are controlled by a **central authority** (e.g. individual, company, government, etc.) who provide their **services** deployed within the premisses or through the cloud.
 - The user needs to trust the central entity.
- **Consensus** is a **distributed** and **trustless** form of **agreement** in the process of **transaction verification**. The result of such a architecture is transaction verification on independent **nodes** controlled by an entity.
- The end-user have to trust only **the network design**.



example of a smart home dApp for monetization



centralized



decentralized



Ethereum ledger ⁽¹⁾

- Blockchain technology uses a P2P network.
- The most known distributed ledger is **Bitcoin** which was the first publicly available network using the blockchain technology.
- In the context of this course we will use **Ethereum** ledger that supports smart contracts:
 - allows “programmable” blocks of chain defined as **Smart Contracts** that contains the logic, states and other features similar to classes in object oriented programming languages.

Some of dApp examples in different domains:

- **origin and authenticity tracking** (e.g. food production, medicine, **Everledger**),
- **identity verification** (e.g. **KYC-Chain**),
- **proof of ownership** (e.g. **TrustToken**),
- **economy of things** (e.g. **Blochains**),
- **decentralized prediction markets** (e.g. **PredictIt**),
- **financing of international trade** (e.g. **we.trade**),
- **crowdfunding** (e.g. **WeiFund**),
- **gambling** (e.g. **Edgeless**).



Ethereum Virtual Machine (EVM):

- EVM is an abstract computer machine that relies to stack, similar to [Java Virtual Machine \(JVM\)](#) or [.NET Common Language Runtime \(CLR\)](#). The computer allows running Ethereum applications.
- EVM is protected in a sandbox since the smart contract cannot access to the sources in the network or the file system.
- **In general**, a smart contract can access to the other contracts.

The lifecycle of a smart contract can be captured in the following steps:

1. A smart contract has a defined **goal**, which is achieved by calling functions from at least one entity or his wallet.
2. The wallet owner **instantiates** the smart contract based on the **constructor**.
3. Calling a **function** from at least one wallet in order to achieve a goal.
4. The achieved goal is reflected in the consensus reached, e.g. in the form of written data in the smart contract and/or execution of the asset transfer as a transaction directly via the smart contract.



- **Ethereum Gas:**
 - Gas is the unit of measurement for transaction fees charged on the Ethereum platform. The amount of gas required to execute a transaction depends on the amount of **computing resources** that the EVM consumes during the execution of the transaction.

- Ethereum Gas units:

Unit	Value in Wei	Wei
Wei	1 Wei	1
Kwei (Babbage)	10^3 Wei	1.000
Mwei (Lovelace)	10^6 Wei	1.000.000
Gwei (Shannon)	10^9 Wei	1.000.000.000
Microether (Szabo)	10^{12} Wei	1.000.000.000.000
Milliether (Finney)	10^{15} Wei	1.000.000.000.000.000
Ether	10^{18} Wei	1.000.000.000.000.000.000

- Supported data types:
 - **Solidity value types:** Signed integers, Unsigned integers, Boolean, Addresses, Enums, Bytes
 - **Solidity reference types:** Fixed-size arrays, Dynamic-size arrays, Array members, Byte arrays, String arrays, Structs, Mapping



- **Structure of the contract:**

- **State variable** stores the **state** of the contract. They can be defined with any **supported type** by the Solidity language. Some types, e.g. within Business.sol contract: *owner*, *myContract*, *price*, *minAmmount*.
- **Event** is part of a contract that **interacts with the EVM transaction log** and its call triggers a **forwarding to all clients subscribed to a particular event** (e.g. *Transfer* event in Business.sol contract).
- **Enum** defines its **own custom data type** with a predefined **set of allowed values** (e.g. *FreshJuiceSize* in EnumExample.sol contract).
- **Struct** defines its **own custom data type** (a structure or an object) that **contains variables** where each can be of a different type (e.g. *Book* in Library.sol contract).
- **Function** encapsulates **contract logic** that can be constrained by function modifiers. It has **access to state variables** and **can trigger an event** defined in the contract.
- **Function modifier** allows **changing the behavior** of a function in a **declarative way**. We usually use it **to restrict to a specific input**. A contract can define multiple modifiers that can be applied to multiple functions (e.g. *onlySufficientResources* in Business.sol contract).



- **Base tools:**
- [Remix IDE](#):
 - Remix Online IDE (Web-based DevEnvironment)
 - Remix Desktop IDE (Electron App)
 - Ethereum Remix (VSCode extension)
 - Remixd (CLI tool)
- [WebStorm](#) with installed [Solidity plugin](#)
- [TruffleSuite](#): framework for local compiling, testing and deployment
 - **Ganache**: Tool to run personal Ethereum blockchain which you can use to run tests, execute commands, and inspect state while controlling how the chain operates.

The screenshot shows the Ganache application window. At the top, there's a navigation bar with icons for ACCOUNTS, BLOCKS, TRANSACTIONS, CONTRACTS, EVENTS, and LOGS. Below this is a status bar with various metrics like CURRENT BLOCK, GAS PRICE, GAS LIMIT, HARDFORK, NETWORK ID, RPC SERVER, MINING STATUS, and WORKSPACE. The main area displays a list of accounts. The first account is selected, showing its mnemonic, HD path, address, balance, transaction count, and index.

MNEMONIC		HD PATH	
candy maple cake sugar pudding cream honey rich smooth crumble sweet treat		m/44'/60'/0'/0/account_index	
ADDRESS	BALANCE	TX COUNT	INDEX
0x627306090abaB3A6e1400e9345bC60c78a8BEf57	99.46 ETH	32	0
0xf17f52151EbEF6C7334FAD080c5704D77216b732	100.00 ETH	0	1
0xC5fdf4076b8F3A5357c5E395ab970B5B54098Fef	100.00 ETH	0	2
0x821aEa9a577a9b44299B9c15c88cf3087F3b5544	100.00 ETH	0	3
ADDRESS	BALANCE	TX COUNT	INDEX



- **Development plan**

- We first create a new folder dapp in our working directory:

```
mkdir dapp  
cd dapp
```

- For development, we will use **Truffle Suite** (requirements), which we install using the following command:

```
npm install -g truffle  
truffle --version  
truffle init
```

- The initialization generates the initial structure within the dapp folder:
 - **contracts** ← A folder with Solidity smart contracts that we want to compile and put on the blockchain.
 - **migrations** ← A folder with configuration files to perform the blockchain migration.
 - **test** ← Folder with unit tests (written in Solidity or JavaScript).
 - **truffle-config.js** ← Configuration file of the Truffle project.

