
SOLIDITY
by Red Babel.

1 Ethereum

Ethereum is a platform intended to allow users to easily write decentralized applications (Đapps) that use blockchain technology[1]. A decentralized application is a distributed application like any others (hence composed of multiple parts possibly remote to one another), with the distinguishing trait that each part is individually able to do its job without depending on the other parts. Rather than serving as a front-end for selling or providing a specific service, a Đapp is a tool for people and organizations at different sides of an interaction to come together without any centralized intermediary (which occur in classic client-server solutions). Intermediation functions, such as filtering and identity management[2] are either handled directly by Ethereum or left open for anyone to provide, using tools like internal token systems and reputation systems[3], to ensure that users are offered high-quality (e.g., responsive, trustworthy, available) services.

The Ethereum blockchain can be described as a blockchain **with a built-in programming language**, or as a consensus-based globally-executed virtual machine. The part of the protocol that handles the network internal state and computation is referred to as the Ethereum Virtual Machine (EVM). The EVM can be thought of as a large decentralized computer containing millions of objects, called "accounts", which have the ability to maintain an internal database, execute code and communicate to each other.

The EVM allows code to be verified and executed on the blockchain, ensuring that it will be run the same way on any machine where a party resides. This code is contained in "smart contracts". All nodes process smart contracts to verify the integrity of the contracts and of their outputs. The applications that use smart contracts for processing the data on the EVM are called Decentralized Applications (Đapp). Every time the EVM performs a computation (i.e.: run smart contracts, make transactions), the user of it (of the computation) must pay for that execution. The payment is calculated in Gas. A unit of Gas is paid in Ethereum cryptocurrency[4] (Ether, or, short, ETH).

1.1 Blockchain

In order to understand what Ethereum is, we should make a step back and explain blockchain and how it works. At its heart, a blockchain is a shared database, called a ledger. Much like a bank, the ledgers of simple blockchains keep track of currency (in this case, cryptocurrency) ownership. Unlike a centralized bank, however, everyone (every node) has a copy of the ledger and can verify anyone's accounts. This is the distributed (or decentralized) part of the chain[5]. Each connected device with a copy of the ledger is called a node, see figure 1.

[1] https://ethereumbuilders.gitbooks.io/guide/content/en/what_is_ethereum.html
[2] It addresses the need to ensure appropriate access to resources across heterogeneous technology environments.
[3] A system that allow users to rate each other in online communities in order to build trust through reputation.
[4] https://ethereum.org/ether
[5] For the remainder of the document we will using interchangeably the terms blockchain/chain.

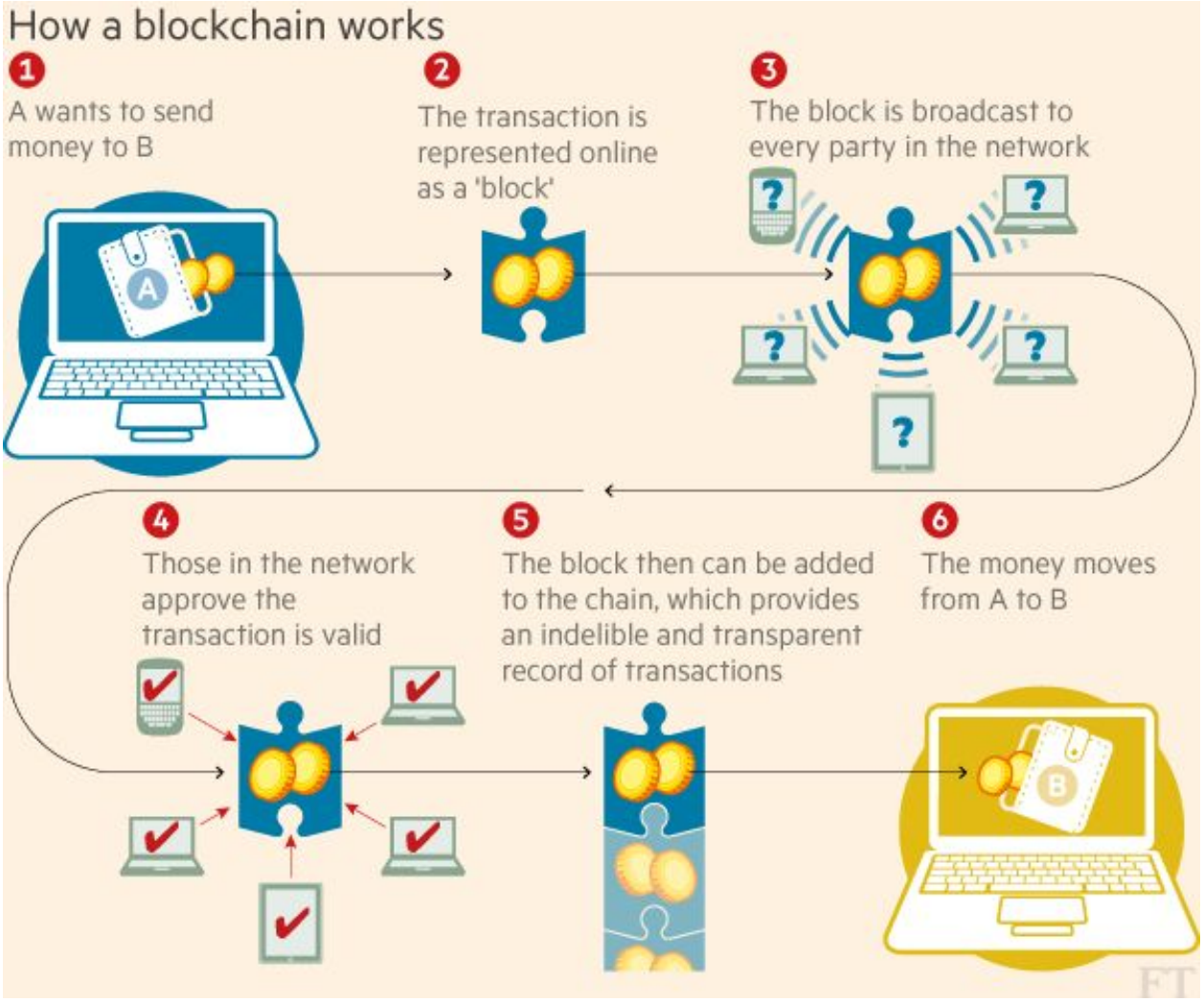

Figure 1: How blockchain works. Source: Financial Times, “Technology: Banks seek the key to blockchain”[6].

Interactions between accounts in a blockchain network are called transactions. They can be either monetary transactions, such as sending someone some Ether, or transmissions of data items, like a comment or username. Every account on the blockchain has a unique signature, which lets everyone know which account initiated the transaction.

Blockchains eliminate the problem of trust with the following key advantages over previous databases:

- Full Decentralization: Reading/writing to the database is completely decentralized and secure. No single person or group controls a blockchain;
- Extreme Fault Tolerance (seen as the ability to handle data corruption): While fault tolerance is not unique to blockchains, they take this ability to its logical extreme of having every party that shares the database to be able to validate its changes;
- Independent Verification: Transactions can be verified individually by any single party, without the need for others. This is sometimes referred to as disintermediation.

[6] https://www.ft.com/content/eb1f8256-7b4b-11e5-a1fe-567b37f80b64

1.2 Smart Contracts

A smart contract is code that runs on the EVM. Smart contracts can accept and store Ether, data, or a combination of both. Then, using the logic programmed into the contract, it can distribute that Ether to other accounts or even other smart contracts. Smart contracts are written in a language called Solidity[7]. Solidity is statically typed, and supports inheritance, libraries, complex user-defined types, and numerous other features.

Figure 2 below shows an example of the working of a smart contract embedding an escrow function. An escrow is a place to store the value of a business transaction/negotiation (e.g. money), until a given condition is fulfilled. In this example, Alice wants to hire Bob to build her a patio, and they are using an escrow contract to store their respective Ether (payment vs. compensation) before the final transaction.

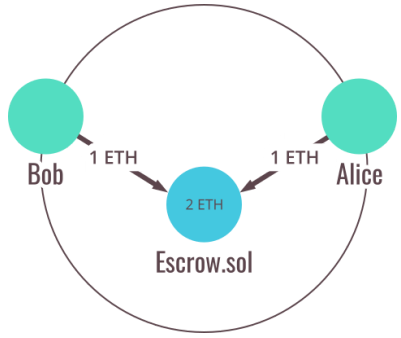

Figure 2.1

1. Alice agrees to store her payment for the patio within the escrow contract, and Bob agrees to deposit an equal amount for compensation (technically called collateral[8]) .

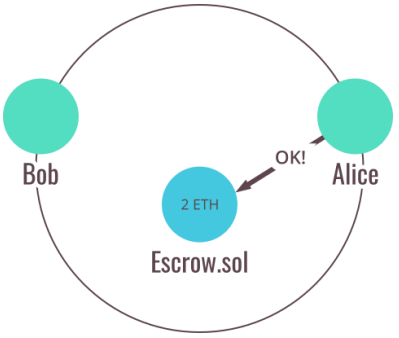

Figure 2.2

2. Bob completes the patio project and Alice gives the smart contract permission to release the funds.

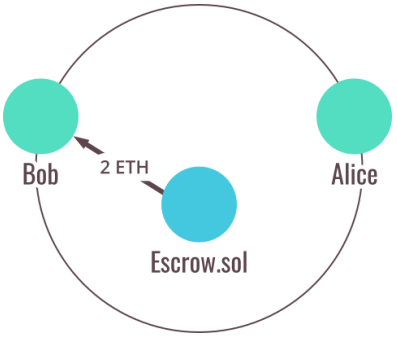

Figure 2.3

3. Bob receives Alice's payment along with his collateral.

1.3 Decentralized applications (Đapps)

Applications that use smart contracts for their processing are called "decentralized applications" (ĐApps). The user interfaces for these ĐApp consist of familiar languages such as HTML, CSS, and JavaScript. The application itself can be hosted on a traditional web server.

1.4 Gas

Ethereum provides the EVM that runs on blockchain. Ether (ETH) is the fuel for the execution of that virtual machine. When you send tokens, interact with a contract, send ETH, or do anything else on the blockchain, you must pay for that operation.
You are paying for the computation, regardless of whether your transaction succeeds or fails. Even if it fails, the node must validate and execute your transaction (compute) and therefore you must pay for that computation just like you would pay for a successful transaction.

[7] https://solidity.readthedocs.io
[8] A value (in this case Ether) pledged as security for the transaction. If Bob were to fail to build the patio then the collateral will be released to Alice. That rule could be written in the smart contract code.

When you hear someone say Gas, the person is either talking about:

- **Gas Limit**
- **Gas Price.**

Typically, if someone just says "*Gas*", they are talking about "*Gas Limit*". You can think of Gas limit as the amount of liters of fuel needed for a car taking a trip. You can think of Gas price as the cost of fuel per liter. Figure 3 summarises the analogy.

A. **Gas price**: If you want to spend less on a transaction, you can do so by lowering the amount you pay per unit of Gas. The price you pay for each unit increases or decreases depending on how quickly your transaction will be included in the ledger.
    a. During normal times (at the moment of writing, see ETH Gas station for real-time info[9]):
        - 20 GWEI, transaction fast execution time (~0.5 min)
        - 1 GWEI transaction average execution time (~1-2 mins)
        - 0.1 GWEI transaction slow execution time (~4-5 mins)

B. **Gas limit**: The Gas limit is called “the limit” because it specifies the maximum amount of units of Gas you are willing to spend on a transaction. This avoids situations where there is an error somewhere in the contract, and you spend 1 ETH....10 ETH....1000 ETH..... going in circles but arriving nowhere. However, the units of Gas necessary for a transaction are already defined by how much code is executed on the blockchain. If you do not want to spend as much on Gas, lowering the Gas limit won't help much. You must include enough Gas to cover the computational resources you use or your transaction will fail due to an *Out of Gas Error*. All unused Gas is refunded to you at the end of a transaction.

	Fuel price/ Gas price	Filled up tank/ Gas limit	Rome-Milan trip/ TX fee	Saved Fuel/ Reimbursed Gas	Total cost
Car	1.5 €/litre	25L	10L	15L	15€
Ethereum	10 GWEI/Gas ($10*10^{-9}$ ETH)	48000 Gas	26000 Gas	22000 Gas	0.00026 ETH

Figure 3. Analogy between gas in Ethereum and fuel in cars.

1.5 ERC 20 Tokens

Ethereum tokens are digital assets that are being built on top of the Ethereum blockchain. They benefit from Ethereum’s current infrastructure instead of developers having to build an entirely new blockchain[10]. ERC20 is a standard interface for tokens. It allows for the implementation of a standard API for tokens within smart contracts. The standard provides basic functionality to transfer tokens, as well as allow tokens to be approved so they can be spent by another on-chain third party[11].
An Ethereum token to be considered ERC20 compliant must implement at least the API defined in the interface[12]. Many implementation are available with different additional capabilities such as the ability to be minted on demand, to have a fixed amount or to be suspended[13].

[9] https://ethgasstation.info
[10] https://blog.coinbase.com/a-beginners-guide-to-ethereum-tokens-fbd5611fe30b
[11] https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md
[12] https://theethereum.wiki/w/index.php/ERC20_Token_Standard
[13] https://github.com/OpenZeppelin/openzeppelin-solidity/tree/master/contracts/token/ERC20

1.6 Ethereum networks

The main Ethereum public blockchain is called **MainNet**, but other networks exist, as anyone can create their own Ethereum network. On MainNet, data on the chain—including account balances and transactions—are public, and anyone can create a node and begin verifying transactions. Ether on this network has a market value and can be exchanged for other cryptocurrency or fiat currencies like Euro.

- **Local test networks**: The Ethereum blockchain can be simulated locally for development. Local test networks process transactions instantly and Ether can be distributed as desired. One good example of local test network is *ganache-cli*[14].
- **Public test networks**: developers use public test networks (or testnet) to test Ethereum applications before final deployment to the main network. Ether on these networks is used for testing purposes only and has no value.
    - **Ropsten**: The official test network, created by The Ethereum Foundation. Its functionality is similar to the MainNet.

1.7 Raiden networks

The Blockchain technology provides a way to find a total order of transactions in a distributed system without relying on a trusted third party. The main advantage of this technology with respect to traditional transition systems is its decentralized nature[15]. The main drawback is the speed of the system. The current version of Ethereum reaches a maximal transaction throughput of approximately 15 transactions per second.

The Raiden Network[16] is an off-chain scaling solution for performing near-instant ERC20 token transfers on the Ethereum blockchain. It enables near-instant, low-fee, scalable, and privacy-preserving payments. It is an infrastructure layer on top of the Ethereum blockchain. The main building block is based on payment channels that allow for unlimited, bidirectional transfers between two participants[17].

[14] https://github.com/trufflesuite/ganache-cli
[15] https://github.com/herrBez/ethereum-scalability/releases/download/1.1.0/ethereum-scalability.pdf
[16] https://raiden.network
[17] https://raiden.network/101.html

2 Concept

Soldino is a platform provided by the Government[18]. It allows business owners to register their business and buy/sell goods/services as well to receive and record taxes. The Government can mint and redistribute to citizen money used to sell and buy goods and pay taxes. Citizens can buy things or services using money minted by the Government. The currency used in this project will be an ECR20 compliant token named *Cubit*.

The goal of the *Soldino* project is to build a set ĐApps running on the EVM. *Soldino* enables the automatic tracking of Value-added tax (VAT). The Government and the Businesses are assisted by *Soldino* to perform the usual accounting operation related to the VAT (e.g. quarterly fill, managing payments, change rates, etc etc). Other types of taxes are out of the scope for this project.

Similarly to the current VAT system, three main actors are involved:

1. Government;
2. Business owner;
3. Citizens.

The business logic encapsulating the interaction between these actors should be written as a set of smart contracts. Each actor will then have a set of well-defined capabilities.

The Government is responsible for creating the money as medium for allowing business transactions and tax payments. The Government mints *Cubit* and gives a certain amount of this medium to the Citizens and the Business Owners. The Government owns the list of the registered businesses. Only the businesses registered can perform transaction in the system.

A Business owner registers its business to the list held by the Government. As soon as the business is registered, it can start selling goods or services. The Business owner, through the business, is able to create the goods or services he/she is selling. On a quarterly basis, the business is entitled to pay the VAT to the government.

Citizens can buy service or goods from businesses using *Cubit*. The Citizens who want to open a business can subscribe to the list and become Business owners.

2.1 Technology requirements and macro architecture

A set of web pages should act as user interface to smart contracts. *Soldino* is therefore composed of two macro modules:

1. Smart contracts: which contain all the smart contracts that must be deployed to the Ethereum network;
2. Web/UI: which contains all the code for rendering the front-end that interacts with the EVM.

The system must be implemented using the Truffle[19] development framework for Ethereum. Truffle helps and guides the development with best practices and tools/utilities that take out most of the boilerplate/routine work. As such, smart contracts compilation, testing framework, deployment of contracts with migrations and, interaction with the frontend is taken care of by it.

[18] For simplicity we conflate Government and tax office.
[19] http://truffleframework.com

Soldino must have a comprehensive set of unit/integration tests and must run in a local environment and at least in the ropsten network.
The above applies equally to both modules.

2.2 Accessibility

The distinctive trait of a ĐApp is its interaction with the Ethereum network. Such interaction means that all the transactions must be signed and paid for in Ether.
This requirement poses two challenges:

1. Access to the Ethereum network;
2. Signing with a private key to allow for the required amount of Ether to be used in the transaction.

The former means that the browser running the ĐApp should have access to an Ethereum node. Of course that exposes the ĐApp to the risk of malicious access (a security concern attached to the management of private keys holding some value, Ether in this case). This risk could be addressed in the Web/UI, but -- owing to the complexity of solving it right --, we leave it out of the scope of this project.

To allow for greater flexibility and better security, we require using Metamask[20], which provides transparent access to the Ethereum network using a public node[21] and includes a secure identity vault, providing a user interface to manage your identities on different sites and sign blockchain transactions.

Finally, we require paying close attention to the following two critical aspects:

1. Deployability of the project by the developer;
2. Accessibility of the project by everybody.

2.3 Environments

A best practice in industry is to divide software development across different environments, Each such environment provides a set of the resources (e.g. networks, servers, file storage, others) needed to run (an instance of) the system. Every environment is capable of running the whole system, but it is entirely independent of the others. Therefore, using one resource in one environment (e. g. store a file or write in a database) does not conflict with the usage of resources in another. Environments are used to test the system before deployment to production. They also allow developers to run the system locally.

In a simple scenario, we have at least 4 environments encompassing the following flow among them.
The developer builds some features and run them in her local computer. When satisfied, she will build a suite of (verification) tests. Such tests could be run locally or as part of a continuous integration [22]process. The features will be then deployed on a staging environment so other people can use/test such features. Finally, the release cycle will determine when the final deployment to *production* will occur.

This project **must** use the following minimal number of environments: Local, Test, Staging, Production.

1. Local: In the local environment, the Ethereum testrpc network provided by Truffle could be used. Truffle provides also a local web server to serve the front end files.

2. Test: The same network and web server could be used for the test environment.

[20] https://metamask.io
[21] https://infura.io
[22] https://docs.microsoft.com/en-us/azure/devops/learn/what-is-continuous-integration

3. Staging: Must be publicly accessible. For it, Ethereum network Ropsten[23] shall be used. Ethers could be found online (e.g. https://faucet.metamask.io). As for the web server, Surge.sh[24] shall used.

4. Production: Not required, but the project should be production-ready. For production, the deployment will be done against the Ethereum main network. Surge.sh shall be used as web server. To interact with the main network, real Ether needs to be used. We will explore the possibility of such deployment in due time.

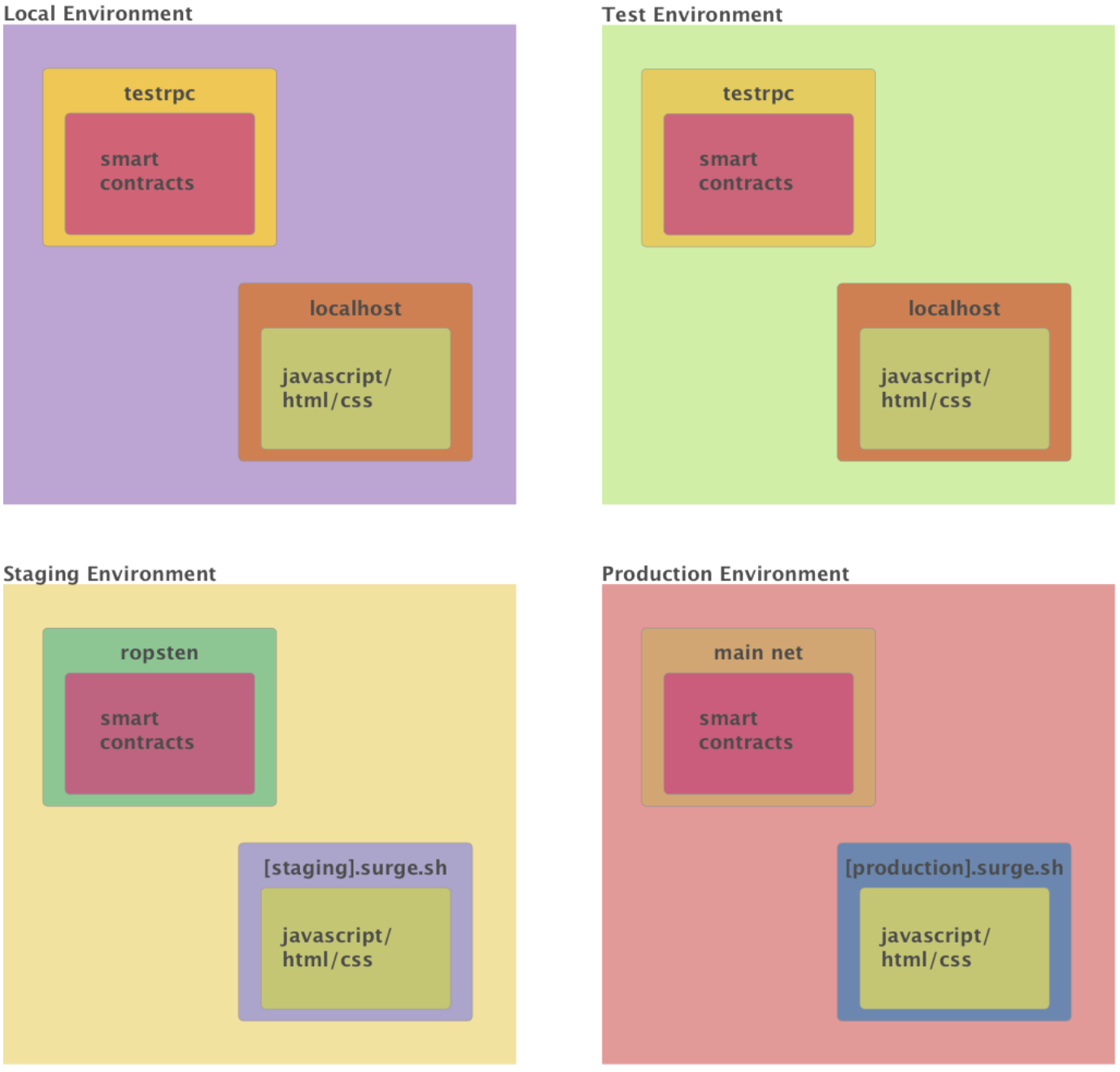

Figure 4. Various environments.

[23] https://ropsten.etherscan.io
[24] https://surge.sh

3 Requirements

3.1 Minimum

Web/UI must provide at least the following list of screens:

1. One or more screens where the the Government can:
    a. Mint and distribuite *Cubit* to Citizens;
    b. Manage the list of the registered businesses;
    c. Check the VAT paid by the business.
2. One or more screens where each Business owner can:
    a. Register the business to the Government list;
    b. Manage (add/remove/update) the goods or services on sell;
    c. Buy goods or services from other businesses;
    d. Create a pdf document with the VAT assessment for the current quarter;
    e. Calculate/Reject/Put on hold/Download the VAT receipt;
    f. Pay the VAT.
3. One or more screens where Citizens can:
    a. Buy goods or services from the businesses;
4. One or more screens where Government and Business owners can:
    a. Signup/Login using Metamask as identity management;
    b. The user cannot perform any action before authenticating against Metamask.

3.2 Optional

To transfer some tokens between parties a transactions must be performed. For security reasons before accepting the transaction as final it is highly suggested to wait for a certain number of subsequent block to be mined (confirmations). As today most websites use between 10 and 30 confirmations where 12 is generally considered acceptable for most use cases. Given the current time for a single block to be mined (~15 seconds) we must assume that a transactions will take around 3 minutes to be completed.
That is acceptable for a set of transactions. In fact wired transfers with the current systems require from few hours (intra country) to a couple of days (intra europe). There is of course a different set of transactions for which 3 minutes waiting is not acceptable (e.g. debit/credit cards).

We want to improve the system as specified so far to allow to perform fast transactions in a similar fashion as using debit card (as opposed to a waiting time of 3 minutes).
Therefore we want to explore Raiden as an off-chain technology to perform fast transactions. A clear model should be specified as part of this requirement on how Raiden would be used as solution.

3.3 Technology

1. *Soldino* must be EIP-712[25] compliant;
2. Smart contracts must be upgradable;
3. *Soldino* will be developed using Javascript 8th edition (ES8) using a promise[26] centric approach. The usage of callbacks must be limited and thoroughly justified (i.e., do not use them);

[25] https://medium.com/metamask/eip712-is-coming-what-to-expect-and-how-to-use-it-bb92fd1a7a26
[26] http://exploringjs.com/es6/ch_promises.html

4. The Airbnb Javascript style guide[27] must be used and enforced using ESLint[28] throughout the development process;
5. It will be developed using React/Redux framework. It is strongly recommended to start from a react/redux boilerplate such as https://github.com/Gigacore/React-Redux-Sass-Starter
6. The usage of SCSS[29] is preferred;

The source code of *Soldino* should be published and versioned using either GitHub or GitLab.
Along with the source code, the necessary documentation should be provided for the end user to use the system and for the developer to run and deploy the modules.

3.2 Warranty and maintenance

The vendor has to demonstrate at the RA *(Revisione di accettazione)* that the product works correctly and according to requirements. Fixing bugs, flaws and any not-compliance with the requirements are entirely at the expenses of the vendor.

3.3 Credits and License

RedBabel holds interests in this project as **proof of concept** of the productivity of the technologies specified above. The system will be distributed under the MIT license, the developer will be mentioned in the copyright credits. Redbabel will be credited to, under the section credits in the README file. Please refer to the Italian law about the management of public bids for what is not specified in this technical specifications document.

3.4 Useful links

- https://blog.zeppelin.solutions: best practices for smart contract development;
- Truffle Framework: development framework for Ethereum;
- Etherscan.io: blockchain explorer. A search engine that allows users to easily lookup, confirm and validate transactions that have taken place on the Ethereum Blockchain;
- Ethgasstation.info: dashboard that shows information about Gas on the ethereum mainnet.
- https://truffleframework.com/tutorials/ethereum-overview;
- https://www.ethnews.com/glossary.

4. The proponent

RedBabel is a Webcraft consulting firm based in Amsterdam and formed by Alessandro Maccagnan and Milo Ertola. The firm operates in Amsterdam and Italy where it holds close relationships with the respective startup ecosystems.
Contacts:

- Alessandro Maccagnan: alessandro@redbabel.com
- Milo Ertola: milo@redbabel.com

To allow for a better and seamless communication between us, the proponent, and you, the vendor, we provide a Slack group available to all group members. To access it: https://slackin-tutcjuiyym.now.sh.

[27] https://github.com/airbnb/javascript
[28] https://github.com/eslint/eslint
[29] https://sass-lang.com