Marvin

by Red Babel.

1 Ethereum

Ethereum is a platform intended to allow users to easily write decentralized applications (Ðapps) that use blockchain technology[1]. A decentralized application is a distributed application like any others (hence composed of multiple parts), but with the distinguishing property 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 Ðapp 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 (Ðapp). 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).

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 blockchain. 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

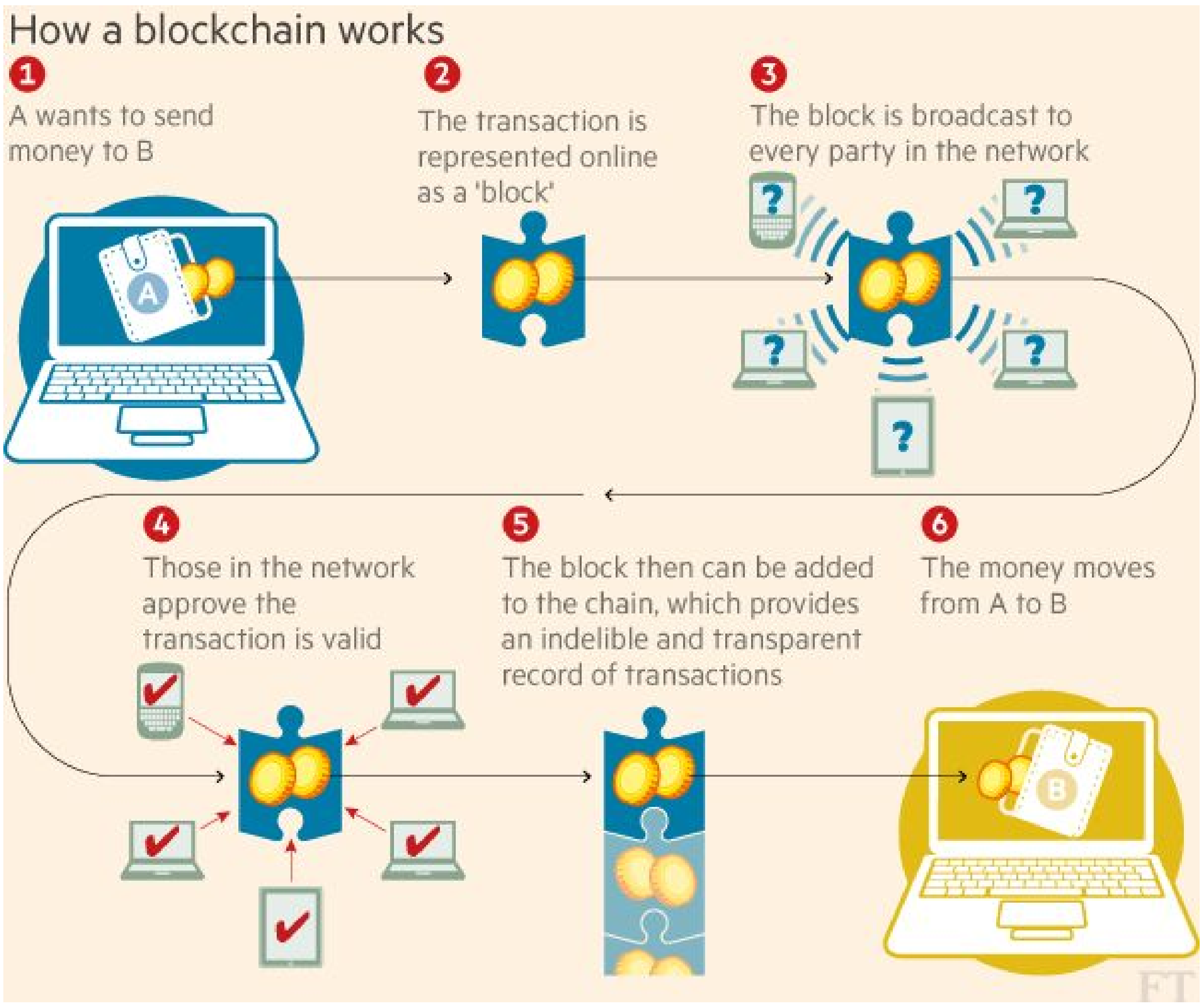

How a blockchain works
1
A wants to send money to B
A
2
The transaction is represented online as a 'block'
3
The block is broadcast to every party in the network
4
Those in the network approve the transaction is valid
5
The block then can be added to the chain, which provides an indelible and transparent record of transactions
6
The money moves from A to B
B
FT

Figure 1: How blockchain works. Source: Financial Times, “Technology: Banks seek the key to blockchain”[5].

Interactions between accounts in a blockchain network are called transactions. They can be monetary transactions, such as sending someone some Ether. Or they are transmissions of data, 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:

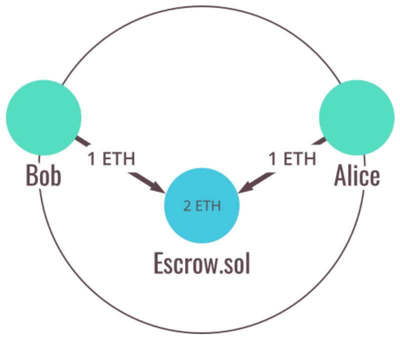
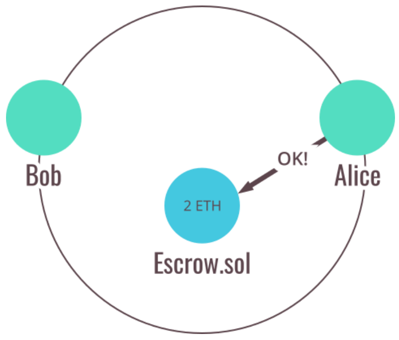
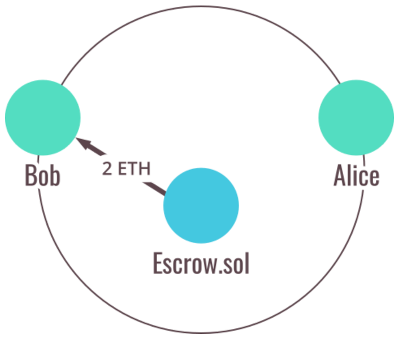
- Full Decentralization: Reading/writing to the database is completely decentralized and secure. No single person or group controls a blockchain;
- Extreme Fault Tolerance (seen as the ability to handle data corruption): While fault tolerance is not unique to blockchains, they take this ability to its 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.

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

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[6]. 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 the transaction/negotiation (e.g. money), until a 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[7]) .	 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.

Decentralized applications (Ðapps)

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

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.

[6] https://solidity.readthedocs.io
[7] 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. You can think of Gas price as the cost the 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 block.
    a. During normal times (at the moment of writing, see ETH Gas station for real-time info[8]):
        - 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.

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 *testrpc*[9].
- **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.

[8] http://ethgasstation.info
[9] https://github.com/ethereumjs/testrpc

2 Concept

Uniweb is a portal provided by the University of Padua. It allows students to access information about their study career and directly manage their university duties, starting from registration, to enrollment of exams, up to final degree application. Professors use Uniweb to publish exam lists, publish votes, and register outcomes[10].

The goal of the Marvin project is to realize a subset of Uniweb functionalities as a ĐApp running on the EVM. Similarly to the Uniweb portal, three main actors are involved in *Marvin*:

1. *University*;
2. *Professors*;
3. *Students*.

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 *University* is responsible for the didactic offer. Every year, it creates a set of degree courses offered to students. A degree course contains a list of exams available for that year. Every exam has a topic, a number of credit points, and an associated professor. The *University* is responsible for which *Professor* is associated with which exam. *Students* can enroll in a specific degree and get a *Libretto* to keep formal track of the progress.
A *Professor* is responsible for running the exam. Specifically, she can register the vote for that exam in the student's *Libretto*.
A *Student* needs to keep track of her progress, which exams she successfully sustained, and which exams he/she have still to pass.

Technology requirements and macro architecture

A set of web pages should act as user interface to smart contracts. *Marvin* 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[11] 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.

Marvin must have a comprehensive set of unit/integration tests and must run in a local environment and at least in one public test network. A final deployment on the MainNet is strongly appreciated as final demonstration, although not required.

The above applies equally to both modules.

[10] http://www.unipd.it/uniweb
[11] http://truffleframework.com

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 therefore require using Metamask[12]. It provides transparent access to the Ethereum network using a public node[13] and includes a secure identity vault, providing a user interface to manage your identities on different sites and sign blockchain transactions.

As a final note, 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.

Environments

A best practice in the industry is to divide the development across different environments, Each of which a set of resources (e. g. networks, servers, file storage, others) needed to run (an instance) 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 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.

3. Staging: Must be publicly accessible. As For it, Ethereum network Ropsten[14] shall be used. Ethers could be found online (e.g. https://faucet.metamask.io). As for the web server, Surge.sh[15] shall used.

[12] https://metamask.io
[13] https://infura.io
[14] https://ropsten.etherscan.io
[15] https://surge.sh

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.

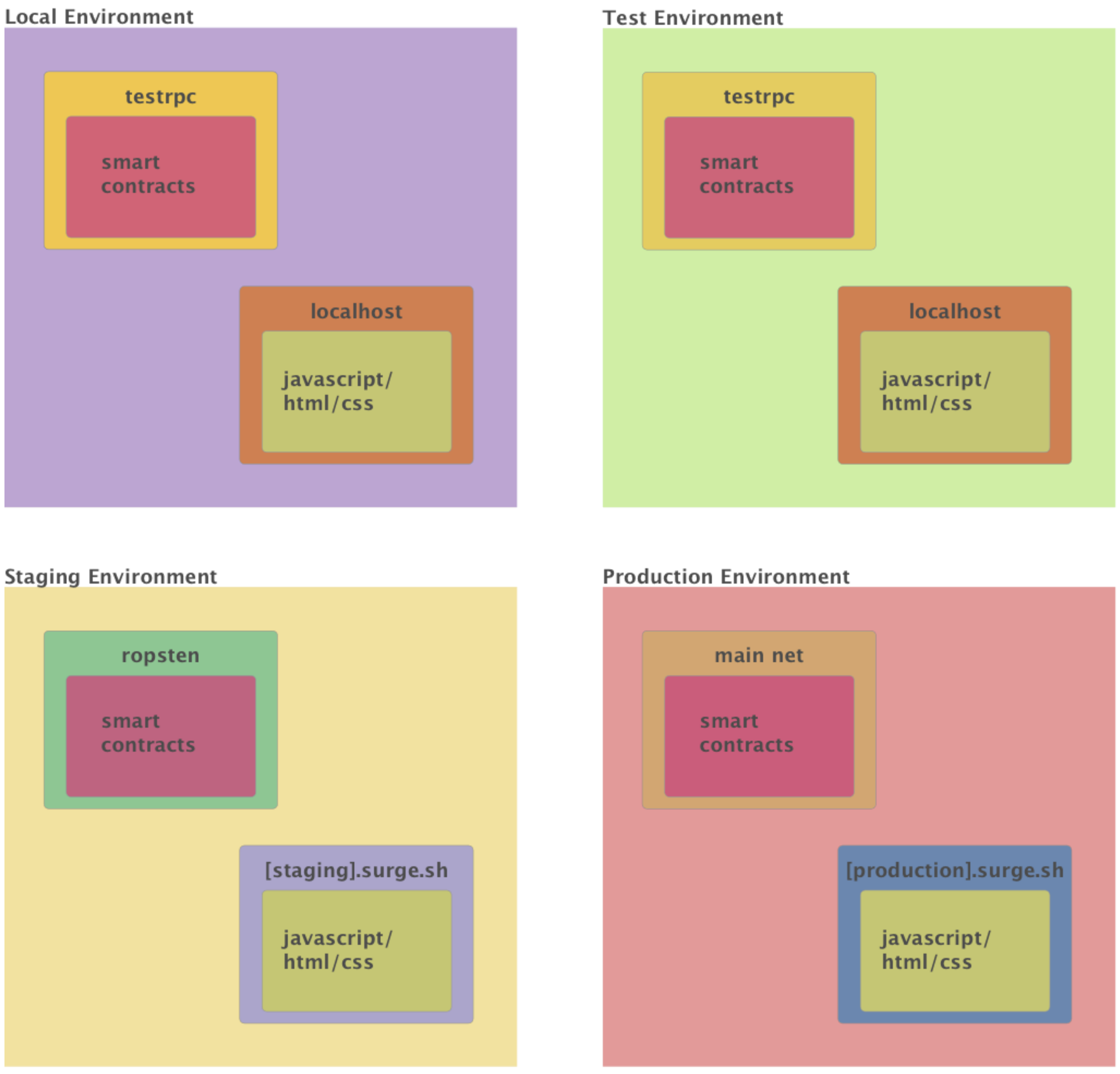


Figure 4. Various environments.

3 Requirements

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

1. One or more screens where the the University can:
    a. Add an Academic Year,
    b. Add a list of professors
    c. For each academic year, the university can add the exams associated to the Academic year
    d. For each exam, the university can associate the professor who can perform the exam registration
2. One or more screens where each Professor can:
    a. See the list of exams associated to him/her
    b. See the list of students associated to each exams
    c. Register the exams to a student associated to that specific exam
3. One or more screens where each Student can:
    a. Choose what exam he/she can subscribe to
    b. See the list of registered exams
4. One or more screens where every user can:
    a. Signup
    b. Login
    c. The user cannot perform any action before the login in the system.

In each page, for every operation the UI must provide a button “Show Me the cost”. The button will show to the user the estimated cost of the related operation. The cost must be shown in:

- Gas
- Ether
- Euro.

Technology

1. Marvin will be developed using Javascript 8th edition (ES8) using a promise[16] centric approach. The usage of callbacks must be limited and thoroughly justified (i.e. do not use them);
2. The Airbnb Javascript style guide[17] must be used and enforced using ESLint[18] throughout the development process;
3. The framework and the demo should follow as close as possible, when relevant, the *12 Factors app* [19] guidelines. The application of the 12 guidelines must be documented;
4. it will be developed using React/Redux framework. It is strongly recommended to start from a react/redux boilerplate such as https://github.com/catalin-luntraru/redux-minimal
5. The usage of SCSS[20] is preferred;

The source code of *Marvin* should be published and versioned using either GitHub, BitBucket, 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.

[16] https://exploringjs.com/es6/ch_promises.html
[17] https://github.com/airbnb/javascript
[18] https://github.com/eslint/eslint
[19] https://12factor.net
[20] http://sass-lang.com

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.

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 too, 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.

Useful links

- 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;
- Etherest.io: exposes API endpoints for anyone to execute a method call or transaction against any smart contract address in the Ethereum network;
- Ethgasstation.info: dashboard that shows information about Gas on the ethereum mainnet.
- http://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.