



8Lab Solutions - Project "Soldino"

# Developer manual

<b>Version</b>	1.0.0
<b>Approval</b>	Samuele Giuliano Piazzetta
<b>Drafting</b>	Francesco Donè Sara Feltrin
<b>Check</b>	Paolo Pozzan
<b>State</b>	Approvato
<b>Use</b>	Esterno
<b>Adressed to</b>	Red Babel 8Lab Solutions Prof. Tullio Vardanega Prof. Riccardo Cardin

## Description

Developer manual made by *8Labs Solutions* for the making of the project *Soldino*.

8labsolutions@gmail.com

## Changelog

Version	Date	Name	Role	Description
2.0.1	2019-03-20	Federico Biciato	<b>RUOLO</b>	Structure of the document created.

## Contents

<b>1</b>	<b>Introduction</b>	<b>6</b>
1.1	Manual contents . . . . .	6
1.2	Purpose of the manual . . . . .	6
1.3	Purpose of the product . . . . .	6
1.4	References . . . . .	6
<b>2</b>	<b>Setup</b>	<b>7</b>
2.1	Requirements . . . . .	7
2.1.1	Browser . . . . .	7
2.1.2	Tools . . . . .	7
2.1.3	Dependencies . . . . .	7
2.2	Installing . . . . .	8
2.2.1	Browser . . . . .	8
2.2.2	Git . . . . .	8
2.2.3	Node . . . . .	8
2.2.4	Truffle . . . . .	9
2.2.5	Ganache . . . . .	9
2.2.6	MetaMask . . . . .	9
2.2.7	Surge . . . . .	9
2.3	Configuration . . . . .	9
2.3.1	Cloning the repo . . . . .	9
2.3.2	Ganache . . . . .	10
2.3.3	Truffle . . . . .	10
2.3.4	MetaMask . . . . .	11
2.4	Running . . . . .	11
2.5	Deploying . . . . .	11
<b>3</b>	<b>React</b>	<b>12</b>
3.1	Overview . . . . .	12
3.2	Components . . . . .	12
3.2.1	Presentational . . . . .	12
3.2.2	Containers . . . . .	12
3.3	Routing . . . . .	12
<b>4</b>	<b>Redux</b>	<b>13</b>
4.1	Overview . . . . .	13
4.2	Unidirectional flux . . . . .	13
4.3	Components . . . . .	13
4.4	How to extend . . . . .	13
<b>5</b>	<b>Web3</b>	<b>14</b>
5.1	Overview . . . . .	14
5.2	Collaborations . . . . .	14

<b>6</b>	<b>Solidity</b>	<b>15</b>
6.1	Architecture overview . . . . .	15
6.2	Contracts . . . . .	16
6.3	Collaborations . . . . .	17
6.4	How to extend . . . . .	18
<b>7</b>	<b>Testing</b>	<b>19</b>
<b>A</b>	<b>Glossary</b>	<b>19</b>

## List of Figures

- 2.3.1 Ganache UI: from top to bottom you can see the menu bar, the current configuration, the mnemonic, and the interface of the selected menu option . . . . . 10
- 6.1.1 Simplified class diagram of the contracts (state variables and methods are excluded) 15

## List of Tables

2.1.1 Packages required for software usage . . . . .	7
2.1.1 Packages required for software usage . . . . .	8
2.1.2 Packages required for development . . . . .	8

# 1 Introduction

## 1.1 Manual contents

This document is the developer manual of the project *Soldino*, developed by *8Lab Solutions* team for the proponent *Red Babel*.

Within the manual you can find:

- the technologies used for the development;
- the software tools used and suggested;
- the software architecture;
- the architectural and design pattern used;
- the functionalities provided by *Soldino*.

## 1.2 Purpose of the manual

The contents of the manual are intended to help the developers who decide to maintain or further develop *Soldino*. Everything described here can help the developer to fully and deeply understand the design, use and features of the application, so that it can be modified and improved with ease.

Many technologies, tools and languages are used to build the app: these are only briefly explained in their parts that cover the application domain. Additional references can be found in the "Reference" section.

## 1.3 Purpose of the product

*Soldino* platform is a DApp accessible on a web browser as a client interface and the plug-in Metamask is a virtual wallet used for transactions made through *Soldino*.

The main functionality of the product is trading goods and services online. Since the platform's backend is based on the Ethereum network, it provides more security and transparency than the traditional e-commerce websites.

The platform is built to be managed by the government and the currency used is called Cubit, it is a ERC20 compliant fork of Ether minted and managed by the government itself.

## 1.4 References

- Ganache <https://truffleframework.com/ganache>
- Git <https://it.atlassian.com/git>
- Node.js <https://nodejs.org/it/>
- Node Package Manager <https://www.npmjs.com/get-npm>
- Surge.sh <https://surge.sh/help/getting-started-with-surge>
- Truffle <https://truffleframework.com/truffle>
- Solidity <https://solidity.readthedocs.io/en/v0.5.0/>

## 2 Setup

### 2.1 Requirements

In this section all of the requirements needed are described.

#### 2.1.1 Browser

*Soldino* is accessible through a web interface. The currently most recent versions of the following browsers are supported:

- **Mozilla Firefox:** version 64 or later;
- **Google Chrome** version 71 or later.

#### 2.1.2 Tools

The following tools are needed:

- **Git:** a famous control version system: *Soldino* is hosted on GitHub;
- **Node.js:** a framework needed for installing dependencies;
- **Truffle:** needed to write and deploy contracts with ease;
- **Ganache:** needed to put up a local Ethereum network and check transactions in it;
- **Metamask:** a browser plugin used as a virtual wallet;
- **Surge.sh:** a web platform chosen for hosting the website interface of *Soldino*.

#### 2.1.3 Dependencies

*Soldino* depends on many different packages, some for use and others for development. All these packages are located in the file `package.json` which is in the root folder of the project. The packages required to execute the software *Soldino* are listed below.

Table 2.1.1: Packages required for software usage

Software	Version
react-text-mask	≥5.4.4
comondir	≥1.0.1
history	≥4.7.2
prop-types	≥15.7.2
react	≥16.8.3
react-dom	≥16.8.3
react-number-format	≥4.0.6
react-redux	≥6.0.1



Table 2.1.1: Packages required for software usage

Software	Version
react-router	$\geq 4.3.1$
react-router-dom	$\geq 4.3.1$
react-router-redux	$\geq 4.0.8$
react-scripts	$\geq 2.1.8$
redux	$\geq 4.0.1$
redux-thunk	$\geq 2.3.0$
web3	1.0.0-beta.37

Other packages, listed below, are required for the development.

Table 2.1.2: Packages required for development

Software	Version
eslint	5.12.0
eslint-config-airbnb	$\geq 17.1.0$
eslint-loader	$\geq 2.1.2$
eslint-plugin-import	$\geq 2.16.0$
eslint-plugin-jsx-a11y	$\geq 6.2.1$
pre-commit	$\geq 1.2.2$
truffle-contract	$\geq 4.0.6$

## 2.2 Installing

### 2.2.1 Browser

The first thing is to have your browser installed. You can get the latest chrome version [here](#).

### 2.2.2 Git

You should type this script in the shell for installing Git packet: `sudo apt install git`.

### 2.2.3 Node

For installing Node.js you have to digit in the shell the following commands:

1. `curl -sL https://deb.nodesource.com/setup_11.x | sudo -E bash -`
2. `sudo apt install -y nodejs`

3. check that `node` have been installed correctly with `node -v`.

There is no need to install npm separately, since it is automatically installed with Node.

#### 2.2.4 Truffle

You should check you have the truffle requirements:

- an OS among Linux, Windows and MacOS (prefer Linux);
- NodeJS v8.9.4 or later (we picked version 11);
- Node Package Manager (npm).

then you can install Truffle by running the command: `npm install -g truffle`

#### 2.2.5 Ganache

There are three step to install Ganache:

1. you can download the Ganache executable at this [link](#), clicking on the download button;
2. if you have a Linux operating system, you have to give the permissions to make the Ganache file executable. This can be done with the command  
`chmod +x path-of-the-appimage/name-of-downloaded-file.AppImage;`

#### 2.2.6 MetaMask

You can add it to your browser in this way:

- Chrome: <https://chrome.google.com/webstore/search/metamask?hl=it>;
- Firefox: <https://addons.mozilla.org/it/firefox/addon/ether-metamask/?src=search>.

#### 2.2.7 Surge

You have to install Surge by executing the shell command: `npm install -g surge`.

### 2.3 Configuration

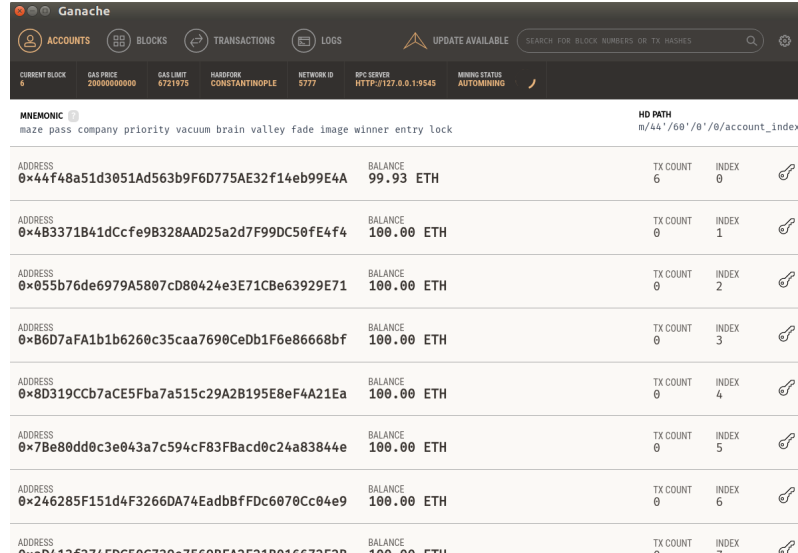
This section shows how to configure your work environment, so that it's the same as ours, in order to minimize the number and entity of troubles you will occur in.

Make sure to configure the tools in order. In particular, Truffle requires that Ganache is opened and is configured to execute successfully.

#### 2.3.1 Cloning the repo

You have to clone the *Soldino* repository on GitHub: open the shell, move to the directory where you want Soldino to be, then use `git clone https://github.com/8LabSolutions/Soldino-PoC`.  
**LINK DA MODIFICARE CON IL LINK ALLA REPOSITORY FINALE**

Figure 2.3.1: Ganache UI: from top to bottom you can see the menu bar, the current configuration, the mnemonic, and the interface of the selected menu option



### 2.3.2 Ganache

You have to go to the folder where you've put Ganache, and open it with double click. Then you have to click on the cog at the top left corner to access Ganache settings and make sure you've matched the following settings (most of which are defined in `truffle-config.js`) on each respective window:

- Server:
  - hostname: 127.0.0.1;
  - port number: 9545;
  - network it: any (you can keep the default one).
- Account & keys:
  - nothing to configure here, but you should have a look at the **mnemonic**: it will help you later.

### 2.3.3 Truffle

The configuration of Truffle is defined in the file `truffle-config.js` in the root directory of *Soldino*. On your shell type the following shell commands:

- `truffle console`  
opens the truffle environment in the shell under the configuration defined in `truffle-config.js`. From now on, every command is executed in the truffle environment, from which you can exit double typing `ctrl + C`.

- **compile**  
to compile the contracts: these are compiled in in a `.json` format (which enables interaction with the frontend) and put into the folder defined in `truffle-config.js` at `contracts_build_directory` (in our case the location is `./src/contracts_build`)
- **migrate --network development**  
puts the contracts running on the blockchain.

#### 2.3.4 MetaMask

It's necessary to create an account.

- open MetaMask on your browser
- select get started
- select import wallet
- use the seed phrase (AKA the mnemonic) copy pasting the one
- from ganache and put your password

Now your account is up and synchronized with Ganache. Now you have to connect the wallet to Ganache network. Ganache settings are exposed in the Ganache UI just above the mnemonic. Let's synchronize MetaMask to the same network. On the top right corner there is a drop-down menu to select the network. Select **Custom RPC** to set your own local network.

You are now in the MetaMask advanced settings screen. In the field **Net Network** click on **Show Advanced Options** to open the form, then insert these data:

1. **New RPC URL:** `http://127.0.0.1:9545` (the port number matches the one in `truffle-config.js`);
2. **Nickname:** the name you wanna give to your network;
3. **save** when you're done.

MetaMask is now connected to Ganache.

To enable transactions on your local test network, you can find free ether for testing networks on this site: <https://faucet.metamask.io/>.

## 2.4 Running

Now that you have all the required software installed and configured, it's time to get it up running.

All you have to do is moving to the root directory of Soldino and prompt

```
npm run start
```

This will open the website on your browser at the address `http://127.0.0.1:3000` and allow you to explore it.

## 2.5 Deploying

This part shows you how to deploy contracts with Truffle and have them up running on Soldino.

## **3 React**

### **3.1 Overview**

### **3.2 Components**

#### **3.2.1 Presentational**

#### **3.2.2 Containers**

### **3.3 Routing**

## 4 Redux

### 4.1 Overview

### 4.2 Unidirectional flux

### 4.3 Components

### 4.4 How to extend

## 5 Web3

### 5.1 Overview

### 5.2 Collaborations

## 6 Solidity

### 6.1 Architecture overview

As you know the main feature of the blockchain is its immutability. It means that once a contract is deployed, it's not possible to modify or update it. In order to update a contract you need to deploy a new contract on the blockchain.

In view of this fact and the proponent's request to develop upgradeable smart contracts, the architecture ideated and developed reflects these two aspects. In fact the smart contracts in Soldino are essentially of three types:

- **Storage contract:** this type of contract isn't upgradeable because it's used to save all the critical data like products, orders, vat movements.
- **Logic contract:** this type of contract implements the business logic. In addition a logic contract as a reference to its storage contract (e.g. UserLogic has a reference to UserStorage). This choice has been made to improve the security of the storage contract (it will be explained in section [DA INSERIRE]).
- **Generic contract:** this type defines the contracts that aren't logic nor storage. These are:
  - **TokenCubit:** contract to implement a custom token ERC20 compliant.
  - **ContractManager:** contract used to achieve simple upgradeability of the system.
  - **Purchase:** contract to allow the user to buy several products with minimum number of transaction.
  - **Owned and Authorizable:** contracts used for the system's security.

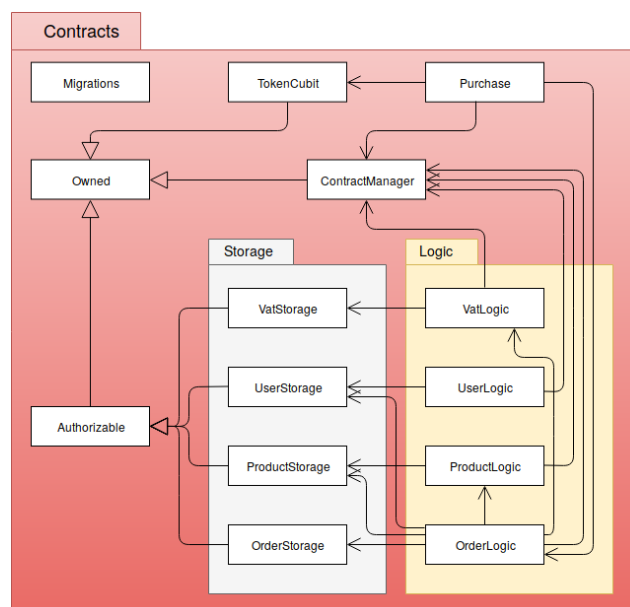


Figure 6.1.1: Simplified class diagram of the contracts (state variables and methods are excluded)



## 6.2 Contracts

## 6.3 Collaborations

## 6.4 How to extend

## 7 Testing

This chapter shows how to

- test the javascript and solidity code automatically;
- check if the code syntax is complied to the rules given.

## A Glossary

A

B

C