

# Blockstarter 4 Report

14.07.2017

Advance Enterprise Computing

Darshan Hingu (380514)

<b>1 Introduction</b>	<b>3</b>
1.1 Requirements	3
<b>2 Blockstarter 4</b>	<b>5</b>
<b>3 Implementation Strategy</b>	<b>9</b>
3.1 Architecture	9
3.2 Solidity Smart Contracts	9
3.3 Blockchain Connector	10
3.3.1 Development Test Setup	11
3.4 Express	12
3.4.1 Design	12
3.4.2 Requirements and setup	13
3.4.3 File Structure	14
<b>4 Possible Extensions and Enhancements</b>	<b>16</b>
4.1 Scalability	16
4.2 Security	16
4.3 Usability	18
<b>References</b>	<b>19</b>

# 1 Introduction

A new concept for building effective and enormously scalable applications is developing. Bitcoin led this initiative back in 2008 with its open-source, peer-to-peer network, cryptographically-storing records (blockchain), and set number of tokens that power the utilization of its components. In the most recent years many applications are following the Bitcoin path. Ethereum being the most promising one, with its decentralized platform that runs smart contracts applications which ensure online contractual agreements. Its tremendous increase of market capitalization has taken a few huge corporations' attention to invest (such as Microsoft) for future developments of "decentralized applications" (Dapp).

In order to be able to understand how blockchain technology works and how the implementation of smart contracts is done through Ethereum platform, first we need to understand what a decentralized application is and what kind of criteria needs to fulfill.

For an application to be considered a Dapp it must meet the following criteria:

- The application must be open-source, it must work autonomously, and with no substance controlling the majority share of its tokens. The application may adjust its protocol in response of proposed improvements and market response however all changes must be done by consensus/agreement of its clients.
- The application's data and records of operation must be cryptographically stored in an open, decentralized blockchain keeping in mind the end goal to maintain a strategy of avoiding any network central operation or bottlenecks.
- The application must utilize a cryptographic token (token local to its framework) which is fundamental for access to the application and any commitment of significant worth from (miners) must be rewarded in the application's tokens.
- The application should generate tokens as indicated by a standard cryptographic algorithm which would act as a proof of the value nodes are adding to the application (Bitcoin and Ethereum currently using the Proof of Work Algorithm) [1].

The aim of this report is describing the development, challenges, implementation strategy and further enhancements and improvements of a Dapp ("Blockstarter") based on the idea of "kickstarter.com" in ethereum environment. Moreover, through this report we provide justified development decisions that were made throughout the project and briefly explaining scalability and security aspects.

## 1.1 Requirements

Any project must have prerequisites/requirements that characterize what the final outcome of the project is at last expected to do. Requirements and specifications are guidelines describing what kind of functions the project will provide, what qualities the product should have, and

what objectives the product should meet. The development of “Blockstarter 4.0” prototype has been done based on the requirements given.

Front-end that	Backend + Smart Contracts that
Allows users (blockchain address) to create new projects	Allows backers to retrieve a share (token) for each project they invest in
Allows project owners to list their projects with funding status	Allows project owners to withdraw funds when a project has been funded successfully (and not be able to withdraw if the funding goal has not been reached, yet)
Allows project owners to cancel a project and refund all backers.	
Allows backers to invest in a project	
Allows backers to view in which projects they have invested	

**Table 1.** Requirements and Specifications

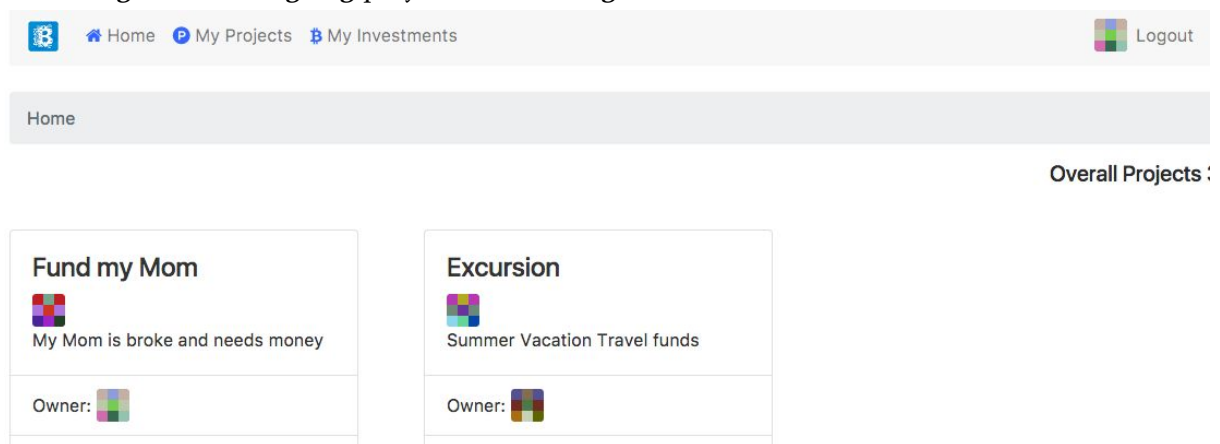
In order to achieve the above mentioned requirements, we decided to split our group in two sub-development teams: Back-end team (blockchain; smart contracts) and front-end team (Express and UI-handlebars). Each team member is involved in developing different components of the project. In one hand, back-end team would be working on smart contracts implementation and establishing a connection of smart contracts to interact with UI. On the other hand, Front-end team would be working on establishing routing, functionalities that would be able to easily display and interact on the UI itself. This implementation strategy had to be organized within the team. Our approach regarding project management is based on the Scrum methodology. We decided to use Taiga project management tool. As a requirement for the whole project development we had three update meetings therefore, we decided to split the project into “sprints” which are basically milestones and assigning tasks via Taiga. Moreover, each team member decides which task/issue she/he believes that they can achieve. We managed to keep our team coordination by having regular team meetings once a week and every two days a short slack meeting by giving a short update on our daily developments.

## 2 Blockstarter 4

### Functionality:

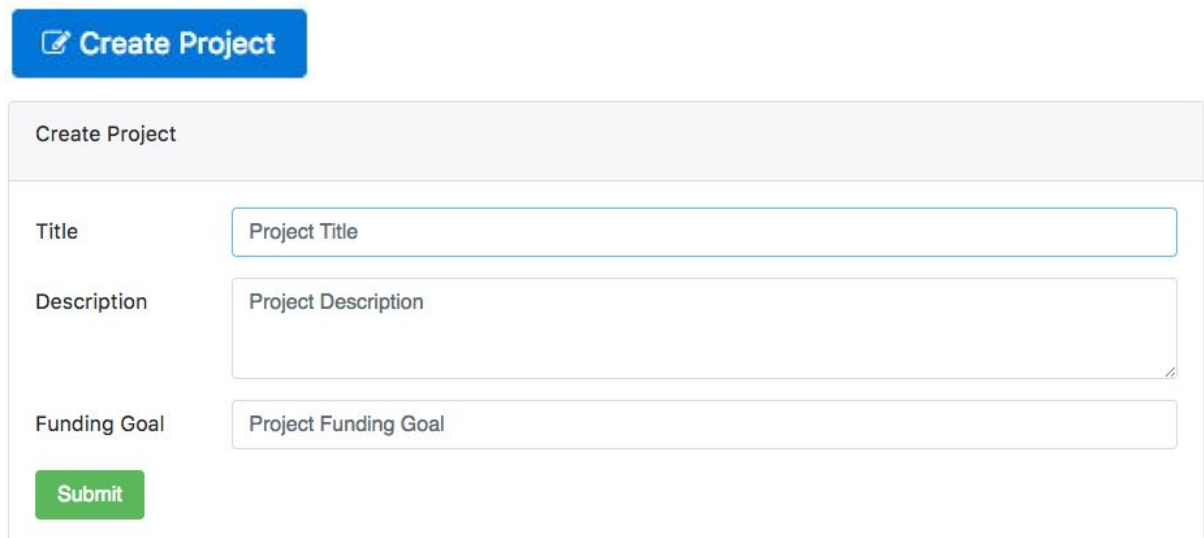
- ❑ The user can login to Blockstarter page with the address.
- ❑ The 'Home' tab lists all the ongoing project tiles.
- ❑ The 'My projects' tab lists the projects owned by the user.
- ❑ The 'My Investments' tab the user can see the projects he has invested in.

Home Page: All the ongoing projects under the global contract are listed here.



The screenshot shows the Home Page of Blockstarter. At the top, there is a navigation bar with a Bitcoin icon, a Home link, a My Projects link, and a My Investments link. On the right side of the navigation bar is a Logout button. Below the navigation bar, there is a section titled "Overall Projects :". Under this section, there are two project tiles. The first tile is titled "Fund my Mom" and has a description "My Mom is broke and needs money". The second tile is titled "Excursion" and has a description "Summer Vacation Travel funds". Both tiles show an "Owner:" field with a small profile picture icon.

**Create Project:** The user can navigate to 'My Projects' tab and create a new project:



The screenshot shows the "Create Project" form. At the top, there is a blue button with a pencil icon and the text "Create Project". Below this button, there is a form titled "Create Project". The form has three input fields: "Title" with the placeholder text "Project Title", "Description" with the placeholder text "Project Description", and "Funding Goal" with the placeholder text "Project Funding Goal". At the bottom of the form, there is a green "Submit" button.

**List Projects:** Under My Projects tab, project owner can list all his projects with Funding Status.

[Create Project](#)



**Fund my Mom**  
My Mom is broke and needs money  
Stage **Funding**  
Current Funding: 3560  
Required Funding: 6000  
Funding **PENDING**

**End Funding / Cancel & Refund:** The project owner can

- ❑ *End funding* for his project once the goal is reached. After Ending the funding, the funding amount is credited to owner's balance.
- ❑ *Cancel* the project and *refund* the money to investors at any time. On doing so, all the investors get their invested amount back.
- ❑ Start a *poll*. This posts a poll for all the investors of the project.

**Stage** **Funding**  
  
Current Funding: 400  
  
Required Funding: 300  
  
Funding **SUCCESSFULLY**  
  
  
  
**Start poll**  
  
**Cancel & Refund**  
  
**End Funding**

**My Investments:** Under this tab, investors can see all their backed Projects.

 [Home](#) [My Projects](#) [My Investments](#)  Logout

Home / My Investments

**Fund my Mom**  
My Mom is broke and needs money

Stage **Funding**

Current Funding: 3560

Required Funding: 6000

Funding **PENDING**

**Excursion**  
Summer Vacation Travel funds

Stage **Funding**

Current Funding: 1810

Required Funding: 2000

Funding **PENDING**

**Invest in a Project:** User can *invest* in a project with an Ether amount. When the Current Funding amount exceeds Required Funding, the Project's funding status flips to *Successful*.


*Invest in a Project:*

Stage **Funding**

Current Funding: 3560

Required Funding: 6000

Funding **PENDING**

\$ 1000| 

Invest

*Successfully Funded:*

Stage **Funding**

Current Funding: 400

Required Funding: 300

Funding **SUCCESSFULLY**

\$ Ether Value

Invest

Extension:

**Tokens:** The investors receive *shares* as tokens upon investing in a project. They can also *transfer* the tokens to other users.

Stage **Funding**

Current Funding: 560


Required Funding: 6000

Funding **PENDING**

Token: 60



Transfer Token

Transfer Token to other user



Submit

**Voting:** The investors have the *voting* rights when they receive the tokens. The more they've invested in the project, the more weightage their vote carries.

 [Home](#) [My Projects](#) [My Investments](#)  Logout

Home / [My Investments](#) / Fund my Mom

# Fund my Mom

My Mom is broke and needs money

Poll: Is your mom broke too?

Yes

No

Stage **Funding**

Current Funding: 560

Required Funding: 6000

Funding **PENDING**

Token: 110

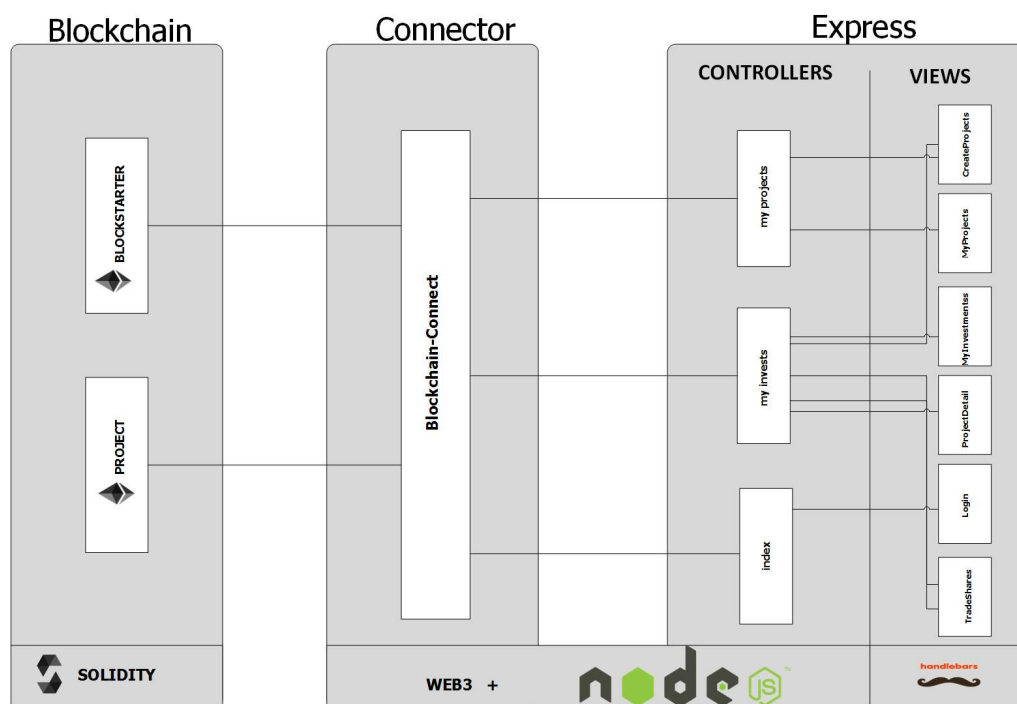
Transfer Token



## 3 Implementation Strategy

In this section our overall architecture will be explained. Especially unusual tools and our approach are covered, and we present disadvantages as well.

### 3.1 Architecture



**Fig 1.** Architecture

### 3.2 Solidity Smart Contracts

Smart contracts in the abstraction level are the same as standard contracts. While a standard contract plots the terms of a relationship (normally by law), a smart contract implements this relationship through cryptography. Smart contracts are programs that are executed precisely as they are written by their makers. Ethereum is a platform that is assembled particularly to create smart contracts. It enables developers to create their own smart contracts. Ethereum Virtual Machine is the runtime for Smart Contracts. It is a virtual machine that keeps running on each node in the network. Moreover, it is isolated that no system or filesystem access thus guarantee determinism. Ethereum Virtual Machine executes these Smart Contracts. However, in order to use for development purposes, higher level programming languages which compile to EVM code have been developed such as Solidity, Serpent, LLL [2].

In our back-end implementation we decided to develop two smart contracts:

- **“Blockstarter”** - being the “global contract” also serving as our decentralized database which holds all the references of projects created on our platform.
- **“Project”** - is the contract which has all business logic behind: the functionality of creating/removing a project, investing in a project, showing the status of the project sharing of tokens, voting rights logic, polls etc.

On every implementation decision over different technologies, there are always trade-offs that are made on the process. Whether implementation of a centralized database or blockchain serving as a “decentralized database” is better; that is moderately debatable. Before the implementation of the project, research is always recommended to be done. During our research on the decision of database implementation, there are advantages and disadvantages that one have to consider.

**Disintermediation:** One of the most important features of a blockchain is empowering a database to be directly shared crosswise over limits of trust, without requiring a central administrator. In the current scope of our project, we believe that with the small amount of data that needs to be stored (small amount of gas needed; therefore small costs) having a “decentralized database” without the need of any maintenance would also be system-cost beneficial for the future developments of our project [3].

**Robustness:** Another important characteristic of a smart contract serving as a database is that is fault tolerance built; which is based on their built-in redundancy. What makes the blockchain as a database special is that it guarantees that nodes which went down can simply catch up on the transactions they missed so they are automatically in sync and no extended configuration is needed[3]. In the context of our project implementation, the robustness gives us an advantage which brings us low cost system and high on redundancy therefore high availability of system achieved

**Performance:** Taking into consideration of the transaction processing done through centralized database vs blockchain, centralized databases are presumably always faster at processing the data. This happens due to the additional three layer process that each blockchain transaction would have to undergo compared to centralized databases: signature verification, consensus mechanism, redundancy transaction processing of independently each node in the network [3]. Especially in our case also queries on the data would have to be implemented performant to keep the platform scaling.

**Confidentiality:** Even though the scope of our project is currently small and is being locally developed, if this would be published on “real-time blockchain”, this might be sensitive to financial aspects. As smart contracts acting that each transaction is visible to every node on the network, many businesses would not accept to share their financial transactions [3].

### 3.3 Blockchain Connector

The blockchain connector is the component that bundles all methods the web server requires to access the model, on the blockchain. All web3 references are therefore centered in this

component and the data requests the web server controllers take are returned by the connector as javascript promises. This simplifies the architecture and decouples the web server from any data storage, as in this case the Ethereum blockchain technology. Furthermore this facilitated the development process, as developers working on different layers of the program just have to decide for a common interface, the connector method signature, and can implement against it.

The current implementation of the payable methods in the connector only accepts unsigned transactions. If the accounts would be locked on the node or the server itself would like to implement signing of transactions, which is controversial, then the connector would have to be modified to implement these additional features.

When loaded the blockchain connector will look for a configuration file containing **abis** and the address of the deployed **blockstarter contract**. If this file is outdated or not available, the connector cannot interact with the blockchain on a network. For our development purposes a tool created this file automatically, as explained in the next section.

### 3.3.1 Development Test Setup

To develop blockstarter 4 an easily usable test environment was required. We tried to use **truffle**, a “development environment for **Ethereum**” [4], but as the setup took longer than we expected, we wrote our own tool. The requirements for such a tool were the following:

- Setup test blockchain the platform can easily connect to
- Provide uri of blockchain node
- If possible, create dummy data as projects on blockchain to enable developers and testers to start immediately
- Provide ABIs
- Deploy **blockstarter contract**
- Provide address of the deployed **blockstarter contract**
- Deliver all required information in a file that can easily be loaded by the platform at startup

Based on **testrpc**, a “Node.js based Ethereum client for testing and development” [10], and our experience from the previous exercise, the blockstarter testrpc tool **bstestrpc.js** was created which executes the following tasks in given order:

1. Compile Contracts
2. Setup testrpc and addresses provided by testrpc
3. Deploy **blockstarter contract**
4. Deploy test data projects, if given
5. Verify if projects were deployed
6. Create config file with all information required by the platform, especially the blockchain connector, to interact with the blockchain part of blockstarter

The dummy projects can be configured directly in the tool itself. After the configuration file was created the web server can be started immediately.

## 3.4 Express

Express is a fast, un-opinionated, minimalist web application framework for Node.js. It is designed for building web applications and is the de facto standard server framework for Node.js. Early 2016 the Express.js project was brought into the Node.js foundation as an incubating project. Express was itself a project in node generation which was split into three Github organizations called expressjs, pillarjs and jshttp. Express is a simple routing plus a sugar layer built on top of the actual base Node.js HTTP server that helps manage a server and the routes. It gives declarative routing without making *switch* statements or *if* statements or big functions, into a basic middleware pattern [5].

One should already be familiar with JavaScript syntax and have Node.js installed to get started. There are many reasons why Express and Node are great choice for server side development. One major reason is that since Express is built on top of Node it is the perfect framework for ultra-fast input and output. Node.js is both asynchronous and single threaded- many requests can be made simultaneously without incurring of bottleneck that would slow down processing. The robust API that ships with express allows us to easily configure routes to send and receive requests from a front-end and connect to a database [5].

In Blockstarter4 our main goal is to implement express as web framework. We would discuss the design of the website, file structure, requirements and setup, controllers and view engine.

### 3.4.1 Design

In this section we would discuss the design of the website. We used bootstrap with handlebars as our main UI design kit. Handlebars is a very powerful templating framework, alongside with bootstrap it gives a lot of flexibility with design. The styling and the javascript component integration made it the best choice for us to use it in our project. Bootstraps modular structure and less complicated stylesheet function, made it quite simpler for us to work on.

Below are the list of pages and its corresponding functionalities:

- **Home Page:** All the projects with its details are listed in tiles. When user clicks on one of the project tiles it shows basic details of that project- Title, Description, Stage, Current Funding, Required Funding and the funding status. Navigation bar in home page only shows two options (i) Home (ii)Login
- **Login Page:** Login/sign-in page appears when user clicks on login link in home page. The required field to login is user's address.
- **Home page after login:** Once the user is logged in, the navigation bar shows more options (i) Home (ii) My Projects (iii) My Investments (iv) logout

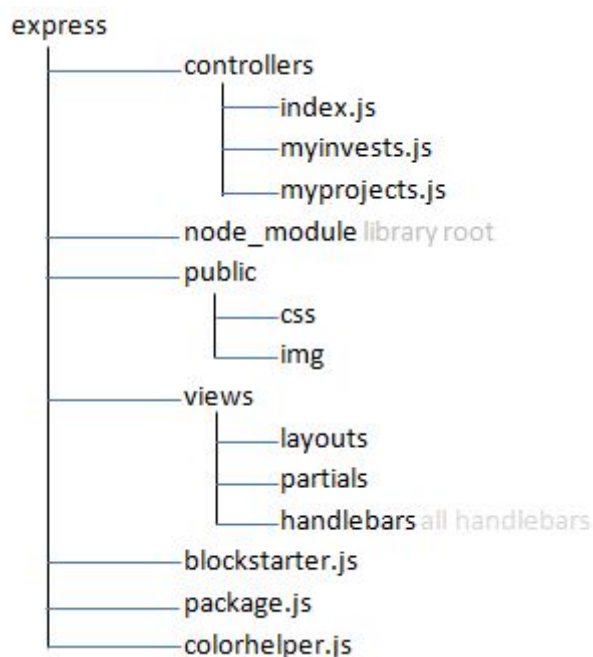
- **Home/My Projects:** By clicking on My Projects user would get a page listing tiles of all the projects that he has created. If no projects were ever created then he can create a project by clicking on Create Project button. User can also delete a project created by him and by clicking on Cancel & Refund button; this deletes the project as well as refunds all the backers/investors who have invested in it.
- **My Projects/Create Project:** The required fields to create a project are: Title, Description and Funding goals. Once all the details are filled and submitted user can see the created project listed in My Projects page; also this project would appear in the home page along with other project tiles.
- **Home/My Investments:** When user clicks on My Investments user would see the list of projects that he has invested in. If no investments are done and user wishes to invest, he has to go back to the home page, click on the project he is interested, enter the Ether and click on *Invest* button. The Current Funding would be increased by the amount invested. This invested project would now appear in My Investments page.
- **Transfer Token:** When user clicks on the project he has invested he can click on Transfer Token button, this enables user to transfer token by entering the address and the token he is wishing to transfer.
- **Start poll:** The project owner can start a poll on the project by entering a question and clicking on *Start Poll*. The investor can participate in the poll as he acquires the voting rights after he gets the tokens. The more he has invested in the project, the more weightage his vote carries. The investor can either vote 'Yes' or 'No' on a poll, and a bar displays the total percentage of positive and negative responses considering the weightage of the individual votes.
- **Logout:** This enables user to logout from his account.

### 3.4.2 Requirements and setup

In this section we discuss about the requirements and setup of Express framework:

- The only thing that is required to be installed is Node.js.
- NPM is also required but it comes along with Node.js package module.
- Generate package.json file- containing all the details about the application and the dependencies to be installed.
- Install express using NPM with save tag.
- The index file in our case is Blockstarter.js, which is used to instantiate and run the application on *localhost* (port being 3000).
  - (i)Blockstarter.js requires 'express' to be imported.
  - (ii)Initialized the express function.
- We chose handlebars as default view engine.  
Default layout for handlebars is set to main.handlebars.
- Used express-session to store user address (its session) and destroying the session when user logs out.

### 3.4.3 File Structure



**Fig 2.** File Structure of Express Framework

Figure 2 represents the file structure that was followed in Express Framework:

- **Controllers:** Controllers are used to declare the routes for all the pages in the website. Every file requires the blockchain file to be included so that the values required could be fetched from them.
  - (i) **index.js:** Contains routes for Home page(/) and Login. These functions get the overall project count, the project list (with details) and displays them.
  - (ii) **myinvests.js:** Contains routes for:
    - '/myinvests'- displaying all the projects that user has invested.
    - '/myinvests/investInProject'- allowing user to invest in a project.
    - '/myinvests/:id/transfertokens'- allowing user to transfer tokens using address and token fields.
  - (iii) **myprojects.js:** Contains routes for:
    - '/myprojects'- gets all the projects owned by the user. Includes a function to view the project when clicked on it.

'/myprojects/createProject'- Allows user to put in details mentioned in previous section and creates a new project. '/myprojects/addProject'- adds the project created to the project list.

'/myprojects/:id/cancel'- It deletes a project owned by the user.

'/myprojects/:address/end'- Ends the funding period of the project.

- **Public:** This folder contains the styling elements for the pages.
  - (i) css: This folder has *custom.css* which is a css file for all handlebars.
  - (ii) img: This folder stores the images that are used in the pages.
- **Views:** Contains “layouts” folder which has “main.handlebars” file which is responsible for our main page. Whatever we see in the GUI on the frontend side is coming from this page. For instance title, body (navigation bar - Home, Login, My Projects, My investments, Logout and the tiles).

It has other files such as -

**404.handlebars** - This is responsible to show the “404 : Not Found” error.

**500.handlebars** - It shows “500 : Server Error”

**createProject.handlebars** - It shows the page when user clicks in “My Projects” to “Create Project” and shows the form to fill to create the project which has the following information - Title, Description and Funding goal.

**detail.handlebars** - Once the project is successfully created it shows the status of the project. For instance- Stage, Current funding, required funding. It shows if the funding is pending or successful and gives an option to invest in the project as well.

**error.handlebars** - This file is responsible for showing us the page with a general error “Something Went Wrong”.

**home.handlebars** - It shows the home-page with the project count and overall projects information.

**investors.handlebars** - It shows “my investments” section with the information such as Stage, current funding, required funding, funding goal etc.

**investorsviews.handlebars** - It shows the additional information about Tokens and Polls.

**listprojects.handlebars** - It shows the list of projects with their details and if none it asks to create it.

**login.handlebars** - It shows the login page where users need to sign in with their unique address.

**transfertokens.handlebars** - It shows the page with the form to transfer tokens to other users and the tokens can be transferred by providing their unique address.

**view.handlebars** - It shows the page with the poll information with additional features like “cancel & refund” and “end funding”.

- **Blockstarter.js:** This file is the backbone of express where we have imported all the required modules like express, fs, express-sessions, handlebars as the template engine, middleware such as body-parser to parse the forms, session to store the session information with the routes and the port information to run the app.
  - **Package.json:** This file contains all the general information about the app like name, version, description etc with all the required dependencies necessary to run the app.



## 4 Possible Extensions and Enhancements

The Blockstart 4.0 Dapp sets with its basic functionality the foundation for a wide range of extensions and enhancements. Since smart contract development is a very young discipline best practices and security patterns haven't yet been established (although currently being developed [14],[15]).

In the following subcategories a broad prospect about how to integrate some of these practices and further extensions to the Blockstarter 4.0 Dapp, to increase scalability, security and usability will be given.

### 4.1 Scalability

Every state modifying function consumes gas. Iterating through an array changes the state by updating the iteration variable. To ensure scalability in the sense of transaction costs and execution time,

- *getAllProjectsForFunder(funder)* and
- *getAllOwnedStatus(owner)*

need to be optimized with further development on Blockstarter.

At the current state every project needs to be examined to perform the functions. That implies for *getAllProjectsForFunder* a complexity of big  $O(n)$  with  $n$  = number of projects, by iterating through the projects array. And for *getAllOwnedStatus* a complexity of  $O(n*m)$  with  $m$  = number of funders for each project, by iterating through every investor of every project. With increasing number of projects and investors this scales badly. Therefore indexing is a solution to this approach of the two functions.

By adding mappings for all projects: *address owner -> address [] projects* and for all fundings *address funder -> address [] projects* of a user to *Blockstarter.sol*, getting all projects from a owner only would consume constant time and has no transaction costs.

### 4.2 Security

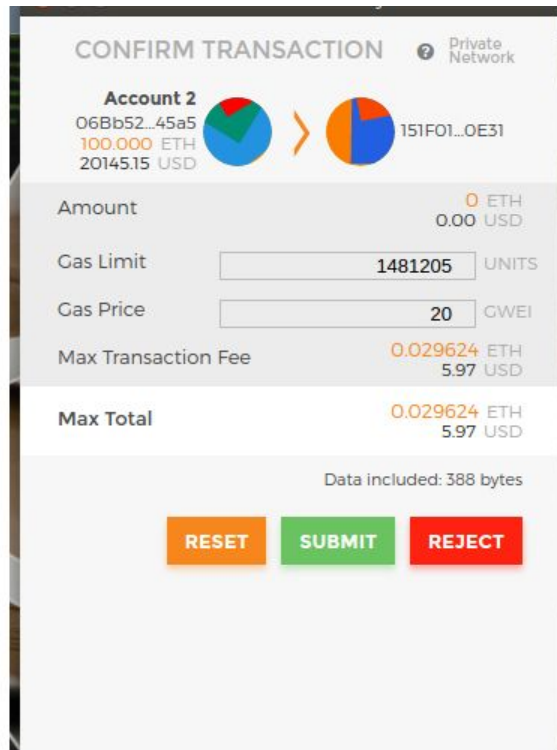
The most important property of a system, especially when dealing with valuable data or assets, is security. Since the security in the Ethereum network is taken as granted, it is important to provide a safe access to the network via Blockstarter 4.0. To do so following topics need to be implemented or improved when further development on Blockstarter 4.0 will be done.

**Unsigned transactions** are not just a security concern also they are not applicable on the real network. Simply because a transaction has to be signed. One way to send signed transactions (e.g. for creating a project) requires manual user interaction, copy & paste. Users need to copy the Blockstarter contract address and the estimated gas, paste that into their Ethereum wallet of choice and send the transaction.

Although this process is very safe, payment process is not handled via the platform, it is inconvenient and therefore will discourage user to use the platform. Another way to interact

with the platform and signing transactions is to use wallet browser integrations. Currently there are three usable implementations:

- **Parity & Mist** : These are full node Ethereum clients which also include a web browser.
- **Meta Mask**: It is a light client Ethereum wallet browser plugin



Being full nodes, Parity and Mist require, depending on the configuration, large amounts ( ~ 10 - 40 Gb) of blockchain data to be downloaded. The setup and configuration of these full nodes is not trivial yet and might also discourage users. Whereas Meta Mask is a light client protocol wallet. Simplified that means only the block headers instead of each full block will be downloaded. [8]

And as a browser plugin (Chrome, Firefox) it is a easy to install tool, which injects the web3.eth transactions and signs them. The above image shows the Meta Mask popup when a Dapp wants to send a transaction. The user then can check on the amount and addresses and submit. Add Meta Mask integration to Blockstarter is a future extension which allows easy Blockstarter wallet interaction and increases the security cause no valuable data is handled via Blockstarter.

**Removing projects** from Blockstarter is currently possible by anyone. One enhancement is it to only allow this to the contract owner. As the 'Project' smart contract provides a owner attribute a simple if condition, *if (msg.sender = project.owner)* in the *remove\_project(address)* function of the Blockstarter contract will do so. Also an administrative account should have the right to remove a project from Blockstarter. The project would still be deployed on its address but just not listed on the platform.

**Login:** currently the platform only uses a identification not an authentication. To ensure the user's identity a login is required. This could be done by simply asking for an email and a

password. It also might be useful to store user email addresses to contact them. But with adding Meta Mask to sign transactions this isn't necessary. The `web3.currentProvider.eth.personal_sign(msg,from)` enables a signature challenge the user needs to confirm with his account. From there on the user can stay logged in with a browser cookie. [11] This also would have the positive effect that user don't have to store any person related data. If this is helpful for a funding campaign needs to be discussed.

A **Minimum Funding** amount would provide not to bloat up the investor array. As explained in 4.1 this might lead to high transaction costs while iterating through all projects. Without the enhancement 4.1 setting a minimum funding amount would prevent this from happening. On the one hand investors can still participate in a project without the platform. But on the other hand one of the many beauties of crypto currencies are the micro payments. Small sums add up to big savings and opens projects and the platform to a wider public. By implementing 4.1 the minimum Funding amount is unnecessary.

**Truffle & Zeppelin:** With further development and the increasing amount of code testing becomes inevitable. Truffle being "the most popular development framework for Ethereum with a mission to make your life a whole lot easier" [12] includes alongside network management, smart contract compilation and deployment automated contract testing.

Zeppelin is a library for writing secure smart contracts which also integrates with truffle [13]. It includes among others multi signature wallets, refundable crowdsale contracts or a time lock token which for example only allow beneficiaries to extract the tokens after a given release time. It also comes with tests for each one of these contract templates.

Since Truffle is build in a modular fashion, which allows only to use certain functionalities, and `blockchain.bstestrpc.js` as mentioned in 3.3.1 fulfills next to the network connection also the contract compilation and deployment, using the Truffle Solidity testing framework and orientating alongside the Zeppelin contract tests will be a guideline for further development.

## 4.3 Usability

By adding Meta Mask as proposed in 4.2 the Blockstart gains usability by enabling fast platform wallet interaction. Also the time consuming registration would disappear. Another beauty of the crypto world is the possible anonymity which will be retained and makes Blockstarter attractive for more user.

# References

- [1] Johnston, David. "The General Theory of Decentralized Applications, Dapps." GitHub. N.p., Feb. 2015. Web. <https://github.com/DavidJohnstonCEO/DecentralizedApplications>
- [2] "Smart Contracts Development." Slides Set 9, n.d. Web. <https://isis.tu-berlin.de/mod/resource/view.php?id=428550> .
- [3] Greenspan, Gideon. "Blockchains vs Centralized Databases." Open Source Private Blockchain Platform. MultiChain © 2017 Coin Sciences Ltd, Mar. 2017. Web. <https://www.multichain.com/blog/2016/03/blockchains-vs-centralized-databases/>
- [4] "ethers/truffle: A development framework for Ethereum", Github, Web. <https://github.com/ethers/truffle> visited on 08.07.2017
- [5] "KEYNOTE: Express, State of the Union by Doug Wilson, Express", Video , Web. [https://www.youtube.com/watch?v=HxGt\\_3F0ULg](https://www.youtube.com/watch?v=HxGt_3F0ULg) visited on 11.07.2017.
- [6] Mist Ethereum Browser, GitHub, Web. <https://github.com/ethereum/mist/releases>, visited on 11.07.2017
- [7] Parity Ethereum client, Web. <https://parity.io/index.html> visited on 11.07.2017
- [8] Ethereum light client protocol, GitHub, Web. <https://github.com/ethereum/wiki/wiki/Light-client-protocol> visited on 11.07.2017
- [9] "What are the Ethereum disk space needs?", Ethereum.StackExchange, Web. <https://ethereum.stackexchange.com/questions/143/what-are-the-ethereum-disk-space-needs> visited 11.07.2017
- [10] "ethereumjs/testrpc: Fast Ethereum RPC client for testing and development", Github, Web. <https://github.com/ethereumjs/testrpc> visited 11.07.2017
- [11] "The New Secure Way To Sign Data In Your Browser", medium.com, Web. <https://medium.com/metamask/the-new-secure-way-to-sign-data-in-your-browser-6af9dd2a1527> visited 11.07.2012

- [12] “Truffle features”, Web. <http://truffleframework.com/> visited on 11.07.2017
- [13] “Welcome to Zeppelin-Solidity”, Web. <http://zeppelin-solidity.readthedocs.io/en/latest/> visited on 11.07.2017
- [14] “Ethereum Contract Security Techniques and Tips”, GitHub, Web. <https://github.com/ConsenSys/smart-contract-best-practices#race-conditions> visited on 11.07.2017
- [15] “Onward with Ethereum Smart Contract Security”, Web. <https://blog.zeppelin.solutions/onward-with-ethereum-smart-contract-security-97a827e47702> visited on 11.07.2017