November 26, 2020

Ollscoil Chathair
Bhaile Átha Cliath
Dublin City University

Trustworthy Crowdfunding Functional Specification

A crowdfunding platform implemented using smart contracts and blockchain technology

Karl Whelan

Student Number: 15561423

Contents

1. Introduction 2
1.1. Overview 2
1.2. Scope 2
1.3. Glossary 3
2. General Description 4
2.1. System Functions 4
2.1.1. Starting a new project 4
2.1.2. Browsing projects in progress 5
2.1.3. Donating to a project 5
2.1.4. Pay funding to creator if goal is reached 5
2.1.5. Return funds to donators if goal is not reached 5
2.1.6. Recording which donators are entitled to rewards 5
2.1.7. Allowing creators to add "stretch goals" 6
2.2. User Characteristics and Objectives 6
2.2.1. Project Creators 6
2.2.2. Project Donators 6
2.3. Operational Scenarios 7
2.4. Constraints 10
2.4.1. Smart contract constraints 10
2.4.2. Storage requirements 10
2.4.3. Testing 10
2.4.4. Time 10
3. Functional Requirements 10
3.1. Create Project 10
3.2. Browse Projects 11
3.3. Donate to Project 11
3.4. Pay-out Funds to Owner 12
3.5. Return funds to Donator 12
3.6. Store Rewards Details for Creator 12
3.7. Expand Scope of Project and Funding Goal 13
4. System Architecture 14
4.1. System Architecture Diagram 14
4.2. Tech Stack Diagram 16
5. High-Level Design 17

5.1. Context Diagram 17
5.2. Use Case Sequence Diagrams 18
5.2.1. Create Project 18
5.2.2. Donate to Project 19
5.2.3. Pay Out or Return Funds 20
6. Preliminary Schedule 20
6.1. Gantt Chart 21
7. Appendices 22

1. Introduction

1.1. Overview

Crowdfunding is currently a popular way for creators to raise money to finance projects. There is a trust issue with crowdfunding though. Most crowdfunding projects do not deliver on time and some never deliver at all. Every crowdfunding platform also charges fees, between 3 and 5% of all raised funding, and they act as a middleman you must trust to transfer your donated funds appropriately.

This system will be a decentralised application, implemented using blockchain technology and smart contracts, that will act as a crowdfunding platform to allow creators to start projects and allow backers to invest in those projects.

This is a good solution to the problems with crowdfunding due to how traceable and transparent each donation and money transfer is on a blockchain. It also removes the need for a middleman as smart contracts with predefined rules can dictate how and when it is appropriate to release funds.

1.2. Scope

The application will implement a rewards-based model of crowdfunding similar to Kickstarter. This means project creators can start a project by defining the features of the project, the funding goal of the project and a time period in which to reach the funding goal. They can also set rewards for different donation sizes. Funds are only paid to the creator at the end of the defined time period, if the funding goal has been reached. If the funding goals of the project are exceeded the creator has the option to add more features to the project and increase the funding goal. Non creators will be able to browse the list of projects in progress and donate to them if they wish. If time permits the scope of the project could be expanded to include other models of crowdfunding.

The systems functionality will come from the execution of smart contracts stored on the Ethereum blockchain. Users will interact with the smart contracts via a geth node and a web-based frontend which can interact with that node. This will allow them to create, browse and donate to projects with the complexity of the blockchain abstracted from them. The frontend will be built using HTML, CSS and JavaScript and interact with the geth node using web3.js which is a JavaScript library. Smart contracts will be written and compiled using Solidity. Safe development and testing will be done using the Truffle framework and Ganache local blockchain. The Rinkeby test blockchain will be used as essentially a staging server.

1.3. Glossary

- **Blockchain:** A decentralised database consisting of blocks that are linked using cryptography. Secure by design. Typically used as a distributed ledger, managed by a peer-to-peer network adhering to a protocol for block validation and node communication.

- **Smart contracts:** A self-executing computer program which implements the terms of an agreement between 2 parties.

- **Decentralised Application:** An application that runs on a distributed computer system. In this case the Ethereum Virtual Machine.

- **Ethereum:** An open source blockchain that features smart contract support. Also, the name of the protocol(s) used for block validation and node communication on this blockchain.

- **Ethereum Virtual Machine:** A software platform running on the Ethereum blockchain. It is a distributed state machine.

- **Eth:** Short for Ether. This is the cryptocurrency built on top of Ethereum. It acts as “fuel” allowing smart contracts to run.

- **Geth:** Short for Go Ethereum. It is a client that implements the Ethereum protocol allowing a machine to become a node on the Ethereum network, validate blocks and interact with other nodes.

- **Web3.js:** A JavaScript API that allows you to interact with a local or remote Ethereum node using HTTP.

- **Solidity:** A programming language for writing smart contracts.
- **Truffle:** A development environment, testing framework and asset pipeline for blockchain development.
- **Ganache:** A personal blockchain that can be run locally to test smart contract functionality and deployment.
- **Rinkeby:** A public test blockchain where are currency used for transactions is fake.
- **Account Balance:** In Ethereum all accounts have an Eth balance associated with them. This includes smart contracts which can also sore Eth.
- **Light node:** Unlike a full node which stores the whole blockchain this type of node only stores the header chain and requests everything else. It allows users to interact with other nodes on the blockchain without the large SSD storage requirements of running a full node.
- **Stretch goals:** In the context of crowdfunding this refers to a funding target that is set beyond the original target for the project. Each "stretch goal" will coincide with an expansion of the scope of the project to add improvements and/or features.
- **Gas:** Refers to a fee paid to miners as an incentive for them to mine blocks. Mining blocks completes transactions and executes contracts.

2. General Description

2.1. System Functions

2.1.1. Starting a new project

The application front-end will allow project creators to start a new project by navigating to the "Create Project" page. Here they will be instructed to enter the attributes of the project which will include a project name, description, funding goal and end date (the date at which the funding goal must be reached). They will also be instructed to set rewards for donation sizes. The front-end will then send a transaction to the address of the smart contract (already deployed on the blockchain) using gas from the creators account. The smart contract will then store all the relevant information on the blockchain.

2.1.2. Browsing projects in progress

The front-end will also allow users to browse the projects that are currently in progress by navigating to the "Explore Projects" page. This page will display the list of projects and their current status which includes all the information entered by the project creator, the rewards for donation sizes, the amount that has been raised so far and the time remaining for the project to reach its goal.

2.1.3. Donating to a project

From the "Explore Projects" page a user can also chose to donate to a project. To do this they will first select the project, then enter the desired amount and click a button. The front-end will then send a transaction to the address of the smart contract using Eth from the donators account which will update the amount raised if there is time remaining for the project. The Eth will be stored in the smart contracts account balance.

2.1.4. Pay funding to creator if goal is reached

If the goal of the project has been reached by the projects end the creator will receive the funds. This will be a function coded into the smart contract that will check if the funding goal has been reached when the time remaining in the project expires. If so, it will send a transaction with the correct amount of Eth to the project creators account.

2.1.5. Return funds to donators if goal is not reached

If the goal of the project is not reached by the projects end the donators will receive their funding back. This will be a function coded into the smart contract that will check if the funding goal has been reached when the time remaining in the project expires. If the goal has not been reached the contract will return the Eth to the accounts of the donators.

2.1.6. Recording which donators are entitled to rewards

The project creator will be able to see which rewards each donator is entitled to based on the amount they donated.

2.1.7. Allowing creators to add “stretch goals”

If the projects funding goal is meet the creator will have the option to expand the scope of the project to include new features or improvements in return for an increased funding goal.

2.2. User Characteristics and Objectives

There will be two main user types for this application, project creators and project donators. Neither type of user will need to have a lot of technical knowledge of software systems as they will be interacting primarily with the front-end which will abstract much of the complexity. They will, however, need to be able to download and install geth and use it to make themselves at least a light node on the blockchain. A simple tutorial of how to do this can be included in the application.

2.2.1. Project Creators

Project creators will be anyone with an idea for a creative project with a clear goal that they need funds to complete. It will allow them to raise funds while also acting as an advertisement to build interest and awareness in their project. The platform would also be suitable for project creators who require investment but don’t want to deal with the risks, costs and other cons of equity or debt investment.

The objectives for these users will be:

- They will want to easily start a new project from the front-end and have that project displayed in a way that is descriptive of what they want to achieve. They will want to have control over the goals, rewards and timeframe of the project.
- They will want to be assured of receiving their funding if the goal of their project is reached.
- They will want to decide weather or not to expand the scope and funding goal of the project if their original goal is exceeded within the original timeframe.
- They will want to control what rewards they offer for certain donation sizes.

2.2.2. Project Donators

Project donators will be anyone who wishes to discover new creative projects that interest them and donate to those projects if they choose. The objectives for these users will be:

- They will want to easily browse projects in progress on the front-end and find projects that interest them.
- They will want to be able to see clearly the features, funding goals, rewards and timeframes of each project.
- They will want to be able to donate to a project if they choose.
- They will want to be assured that they will only lose funds if the funding goal of the project is reached.
- They will want to be assured of getting rewards they are entitled to based on donation sizes.

2.3. Operational Scenarios

Use Case 1	**Create a project**
Goal in context	A project creator wants to start a new project and store it on the blockchain.
Actors	Project creator.
Preconditions	The user is running at least a geth light node and has accessed the application front-end.
Trigger	The user navigates to the “Create Project” page.
Success end conditions	The user receives a message telling them that the project has been successfully created.
Failure end conditions	The user receives a message informing them of the nature of the error.
Main course steps	Step 1: User enters the project attributes in the relevant fields. This includes project name, description, funding goal, end date etc. Step 2: User clicks “Create Project” button. Step 3: A JavaScript function collects the attributes and calls a web3.js function to send these attributes to the address of the smart contract, that is deployed on the blockchain, as a transaction. The gas used is from the creators account. Step 4: A function in the smart contract stores all the relevant information on the blockchain and returns a positive response. Step 5: The user receives a message that the project has been created.
Alternate path steps	Step 1a: If any of the attributes are entered in the wrong format, are empty or are otherwise

	not valid the user is informed of this and returns to Step 1.
Exceptions	Step 3e1: The user does not have an account or enough Eth in their account to send the transaction. The user is informed of this and the function fails. Step3e2: The user is not running a geth node so cannot connect to the blockchain. The user is informed of this and the function fails.

Use Case 2	**Donate to a project**
Goal in context	A project donator wants to donate to a project.
Actors	Project donator.
Preconditions	The user is running at least a geth light node and has accessed the application front-end.
Trigger	The user navigates to the "browse projects" page.
Success end conditions	The user is informed that the donation has been successfully completed.
Failure end conditions	The user is informed that there has been an error.
Main course steps	Step 1: The user selects a project from the list of in progress projects and clicks the donate button. Step 2: The user enters the desired donation amount in the input field and clicks enter. Step 3: The user is prompted to confirm that they wish to donate this amount by clicking yes in a pop-up box. Step 4: A web3.js function is called to send this donation as a transaction to the address of the smart contract. The Eth and gas used come from the donators account. Step 5: A function in the smart contract updates the amount raised for that project and returns a positive response. Step 6: The user is informed that the donation has been successful.
Alternate path steps	Step 2a: The amount entered is invalid or in the wrong format. The user is informed of this and returns to Step 2

	Step 3a: The user selects no in the confirmation pop up box and returns to Step 1.
Exceptions	Step 4e1: The user does not have an account or enough Eth in their account to send the transaction. The user is informed of this and the function fails. Step4e2: The user is not running a geth node so cannot connect to the blockchain. The user is informed of this and the function fails. Step 5e: The project has already ended so the user cannot donate. The transaction fails and the user is informed that the project has ended.

Use Case 3	**Project ends**
Goal in context	The funds raised are paid to the appropriate Actor(s).
Actors	Project creator and project donators.
Preconditions	A project has been created and stored on the blockchain.
Trigger	The project has reached its end date.
Success end conditions	The creator receives the funds raised to their account. OR The donators receive their funds back.
Failure end conditions	None.
Main course steps	Step 1: The smart contract confirms that the amount raised is greater than or equal to the funding goal. Step 2: The smart contract sends the funds raised to the account address of the project creator as a transaction. Step 3: The project ends.
Alternate path steps	Step 1a: The amount raised is less than the funding goal. Proceed to Step 2a. Step 2a: The smart contract returns the funds to the addresses of the donator’s accounts. Proceed to Step 3.

2.4. Constraints

2.4.1. Smart contract constraints

There are several considerations when coding smart contracts compared to other types of programs. The major one is gas limit. Every function in a smart contract costs gas to execute and very block has a limit of gas which can be used in one transaction. This means that functions need to achieve the desired functionality in a computationally efficient. For example, loops that do not have a fixed number of iterations can quickly exceed gas limits.

2.4.2. Storage requirements

For a user to download a geth light node they will need to have around 500MB of Solid-State Drive space. For downloading a full node on the Rinkeby test net blockchain for testing purposes I will need around 50GB+ of SSD space.

2.4.3. Testing

The initial stage of testing functionality and deployment will be done on Ganache, a local blockchain. This is a good testing environment, but it doesn't mimic deploying and functioning the main Ethereum network. Testing on the main network is a very bad idea as it will cost real money in the form of gas. For this reason, it is vital to test extensively in a test network where all Eth and gas used is fake. In order to use a test net though you must download a test net node and request test ether from a faucet. This means the test funds you have to work with are limited.

2.4.4. Time

This is an ambitious project and the limited time for development is a constraint.

3. Functional Requirements

3.1. Create Project

- **Description:** Allows a user to start a new project on the front-end by entering the relevant information. This information is then sent to a geth node via a web3.js API call and then sent to the address of the smart contract that is deployed on the blockchain. The smart contract stores the relevant information on the blockchain.

- **Criticality:** Essential and the first requirement that must be meet for the others to be implemented.
- **Technical Issues:** The attributes of the project need to be stored in an efficient way on the blockchain to avoid unnecessarily high gas usage or exceeding gas limits altogether.
- **Dependencies:** None.

3.2. Browse Projects

- **Description:** Allow a user to browse a list of in progress projects on the front-end. The projects attributes must be clearly displayed including the description of the project, the funding goal, the amount raised so far and the time remaining. The rewards for donation sizes must also be clear.
- **Criticality:** Essential.
- **Technical Issues:** How to access all the projects and their attributes from the front-end without constantly calling a function that uses gas unnecessarily.
- **Dependencies:** 3.1.

3.3. Donate to Project

- **Description:** Allow a user to select a project from the list of in progress projects and donate towards it. The user will enter an amount which will be sent as a transaction of Eth from the donators account to the smart contract. Error handling must clearly convey to the user what the problem was if the transaction cannot be completed e.g. there is not enough Eth in the senders account, the project time period has expired etc.
- **Criticality:** Essential.
- **Technical Issues:** The best way to handle donations should be found. The obvious way is to have the donations paid to the contract itself and then release the funds to the appropriate accounts at the end of every project.
- **Dependencies:** 3.1 and 3.2.

3.4. Pay-out Funds to Owner

- **Description:** When a project ends if the funding goal has been reached all funds should be paid to the creators account. A function in the smart contract can check if the project has ended. If at this point the goal has been reached it will send a transaction to the creators account.
- **Criticality:** Essential.
- **Technical Issues:** The main issue with this is how to initiate the function. Due to the nature of blockchains functions can only be called with transactions so functions cannot periodically call themselves for example. Also testing time dependent logic on blockchains is difficult.
- **Dependencies:** 3.1, 3.2 and 3.3.

3.5. Return funds to Donator

- **Description:** The opposite of 3.4. If the funding goal is not reached the donators should receive their funds back.
- **Criticality:** Essential.
- **Technical Issues:** Same as 3.4.
- **Dependencies:** 3.1, 3.2 and 3.3.

3.6. Store Rewards Details for Creator

- **Description:** The details of which donators accounts are entitled to which rewards needs to be stored in a way so the project creator has access to them or in a way where the smart contract itself can conduct the delivery of the rewards.
- **Criticality:** Essential.
- **Technical Issues:** The issues here will involve the best approach to storing this information and in what format. Storing too much information on the blockchain will cause gas prices to be overly high or will exceed limits altogether.
- **Dependencies:** 3.1, 3.2 and 3.3.

3.7. Expand Scope of Project and Funding Goal

- **Description:** If the funding goal of the project is exceeded with time to spare the creator should have the option to increase the funding goal in return for expanding the scope of the project.

- **Criticality:** Not essential but desirable.

- **Technical Issues:** How this will be implemented needs to be decided for example weather to require the project creators to set out potential stretch goals before they create the project or whether to allow them to update them during the project.

- **Dependencies:** 3.1, 3.2 and 3.3.

4. System Architecture

4.1. System Architecture Diagram

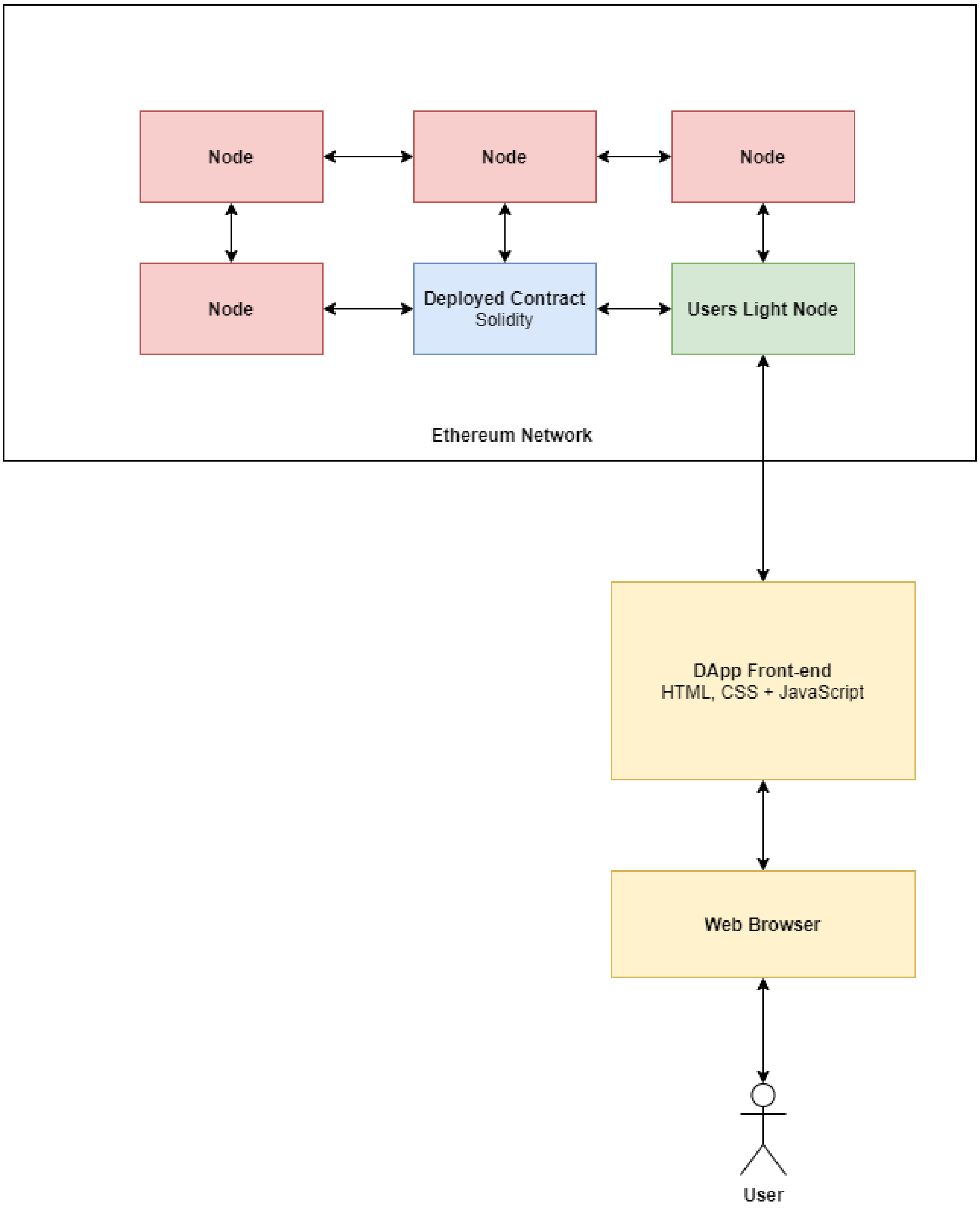


Fig 4.1

Figure 4.1 shows the system architecture. A user interacts with a web browser connected to an instance of the front-end. The front-end will be web-based and written in HTML, CSS and JavaScript. It will allow users to create a project, browse projects in progress or donate to a project. When a function is called it will connect to the users geth node using web3.js calls. This node can then interact will all nodes on the network and will pass the information to the address of the smart contract.

The smart contract will be written in the Solidity programming language, compiled into byte code and deployed on the blockchain. It will provide the functionality needed for the application.

4.2. Tech Stack Diagram

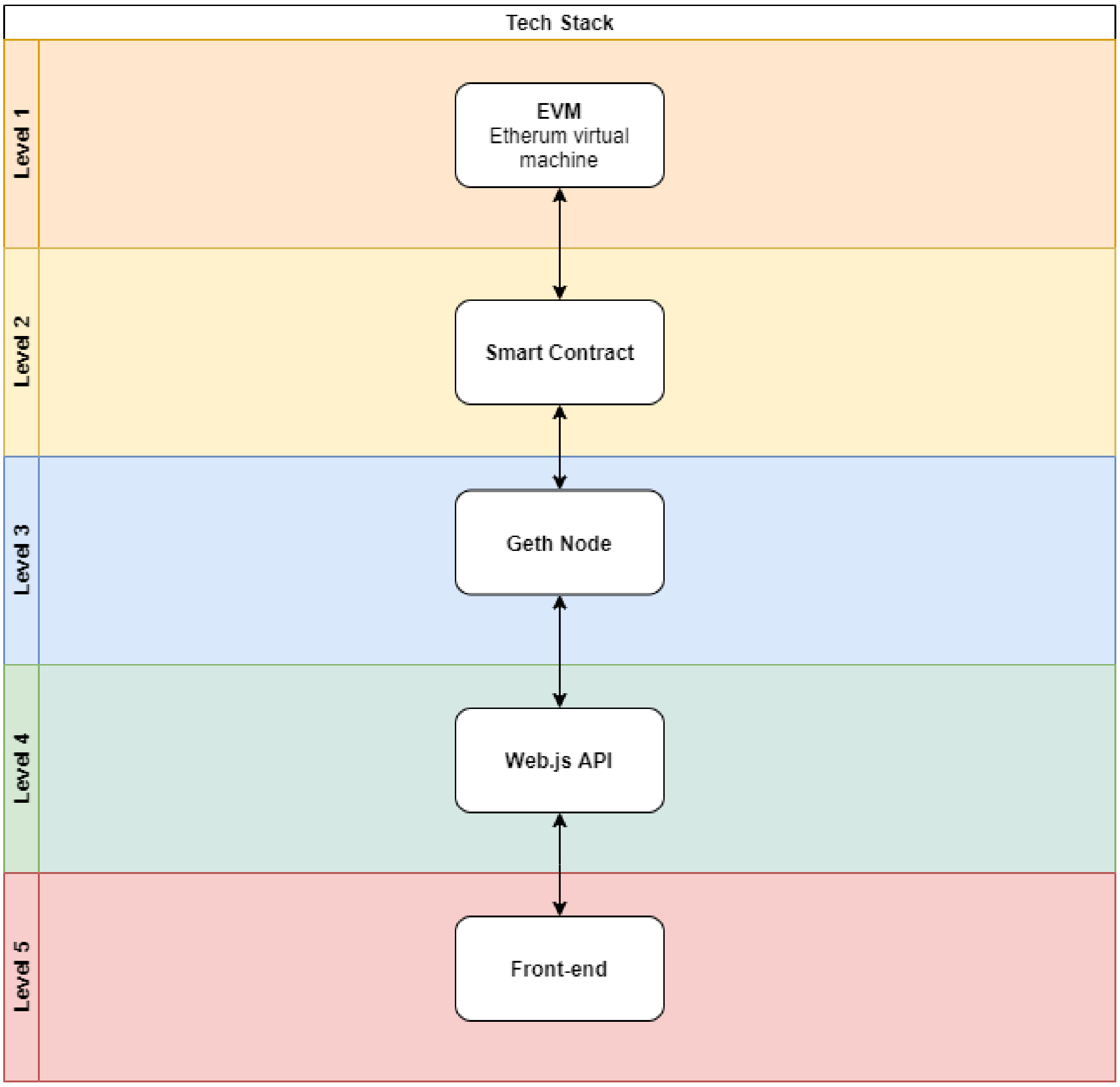


Fig 4.2.

Figure 4.2 shows the tech stack for the decentralised application. The levels are:

- **Level 1:** The Ethereum virtual machine. This is a distributed state machine running on thousands of nodes that accepts bytecode.

- **Level 2:** Smart contracts. There are executable programs written in specific programming languages, in this case solidity. They provide public functions that decentralised applications can interact with. They are compiled into bytecode and run in the EVM.

- **Level 3:** These are Ethereum nodes, in this case a geth node, that are connected to the blockchain and can interact with other nodes. They can send transactions to account addresses and contract addresses.
- **Level 4:** This is client APIs, in this case web3.js. They allow user end application to communicate with geth nodes.
- **Level 5:** The user facing front-end in HTML, CSS and JavaScript.

5. High-Level Design

Below is a context diagram showing the high-level design of the system.

5.1. Context Diagram

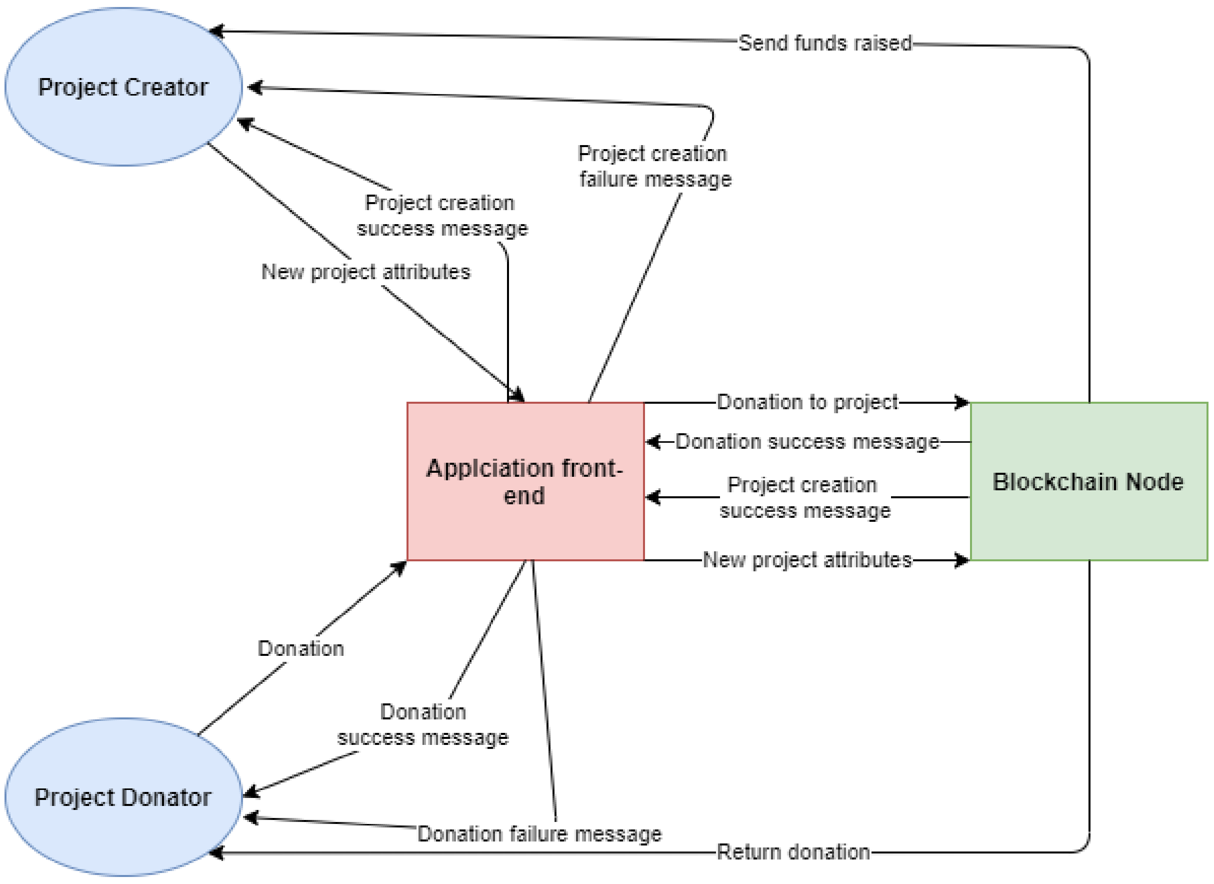


Fig 5.1

5.2. Use Case Sequence Diagrams

5.2.1. Create Project

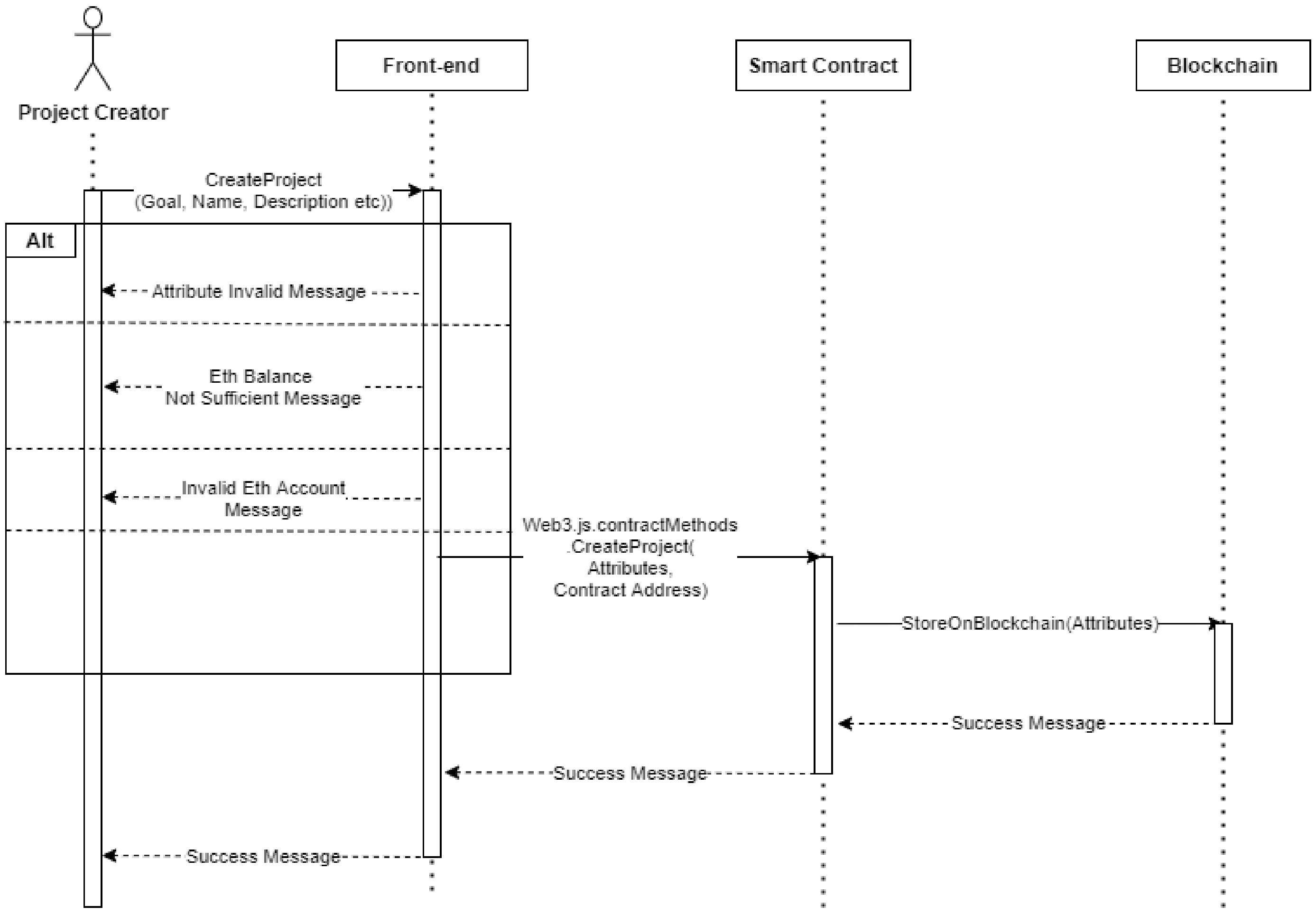

Project Creator
Front-end
Smart Contract
Blockchain
CreateProject (Goal, Name, Description etc))
Alt
Attribute Invalid Message
Eth Balance Not Sufficient Message
Invalid Eth Account Message
Web3.js.contractMethods .CreateProject(Attributes, Contract Address)
StoreOnBlockchain(Attributes)
Success Message
Success Message
Success Message

Fig 5.2.1.

5.2.2. Donate to Project

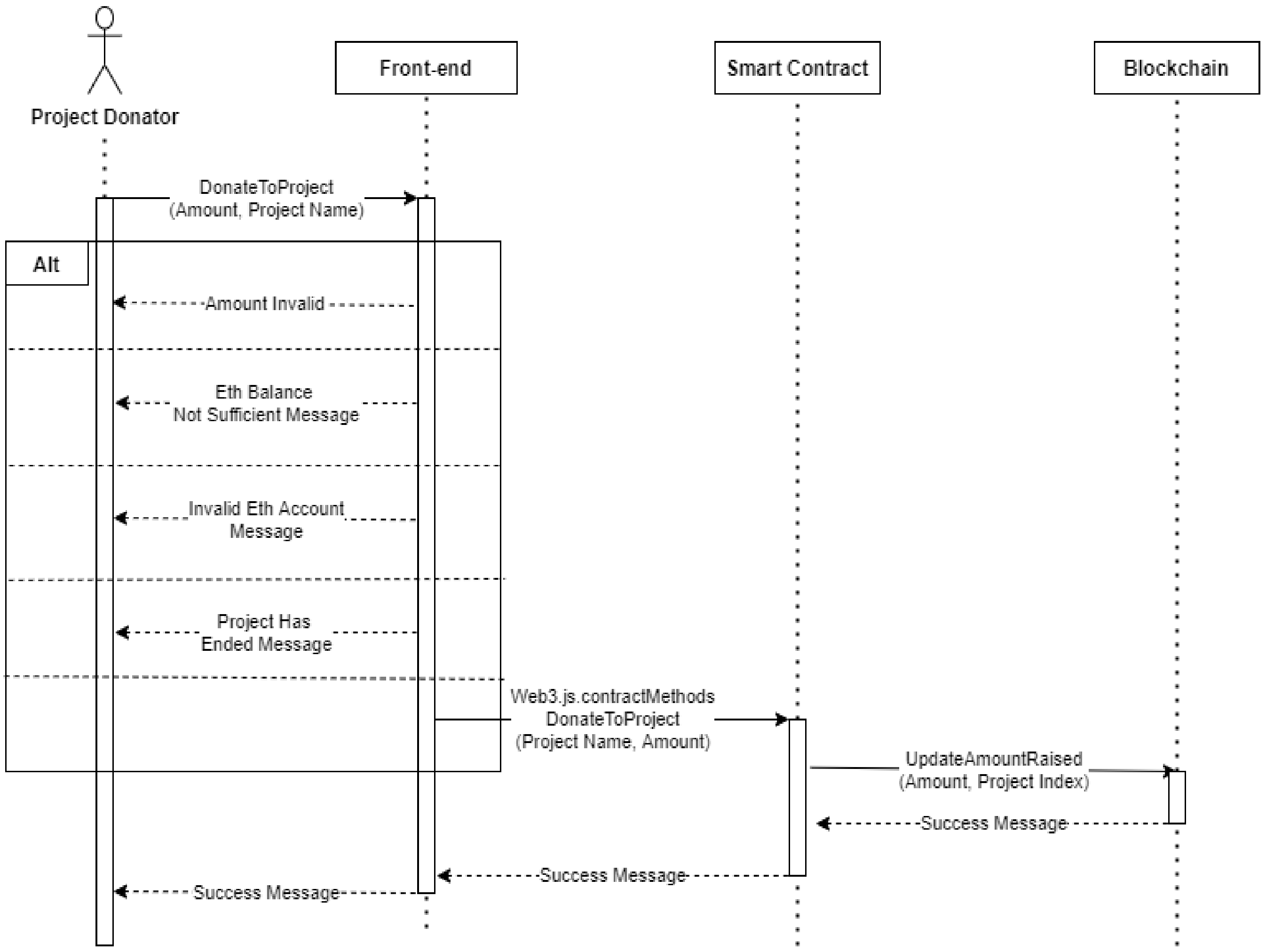
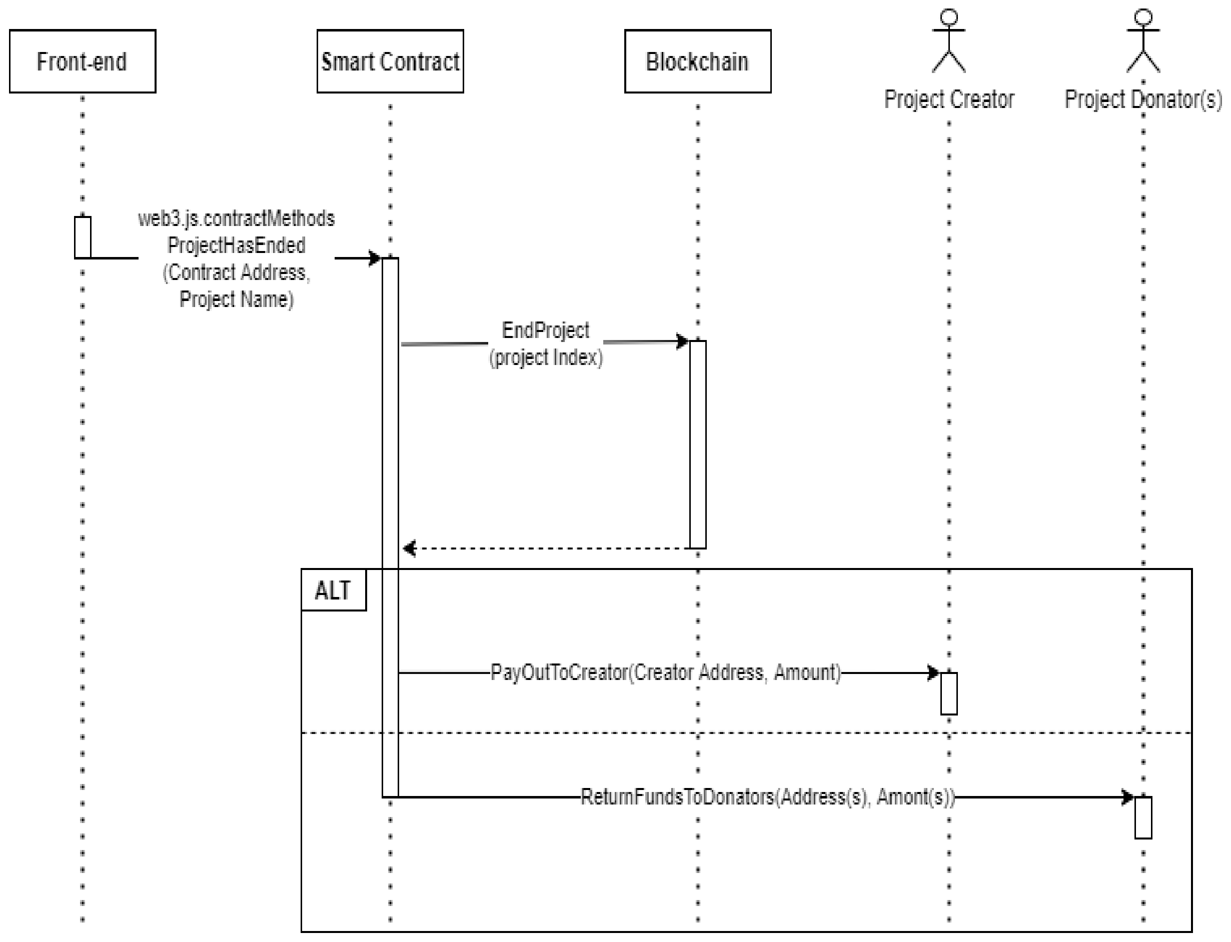

Project Donator
Front-end
Smart Contract
Blockchain
DonateToProject (Amount, Project Name)
Alt
Amount Invalid
Eth Balance Not Sufficient Message
Invalid Eth Account Message
Project Has Ended Message
Web3.js.contractMethods DonateToProject (Project Name, Amount)
UpdateAmountRaised (Amount, Project Index)
Success Message
Success Message
Success Message

Fig 5.2.2.

5.2.3. Pay Out or Return Funds



Front-end
Smart Contract
Blockchain
Project Creator
Project Donator(s)
web3.js.contractMethods ProjectHasEnded (Contract Address, Project Name)
EndProject (project Index)
ALT
PayOutToCreator(Creator Address, Amount)
ReturnFundsToDonators(Address(s), Amont(s))

Fig 5.2.3.

6. Preliminary Schedule

I will use an interactive development approach to this application with each version adding a new function or an improvement of an existing function. Every iteration will have the following steps due to the nature of blockchain development:

- **Step 1:** functionality is added to the smart contract or a new smart contract is developed. Functionality is added to the front-end.
- **Step 2:** functionality and deployment are tested using a Ganache local blockchain.

- **Step 3:** Smart contract is uploaded to the Rinkeby test-net.
- **Step 4:** Further testing is done on the test-net.

The application won't be deployed on the main blockchain until it has all essential functionality and has been thoroughly tested. I will use the Agile software development idea of weekly sprints to iteratively plan and develop functionality and improvements. A Gantt chart showing the sprint schedule is below.

6.1. Gantt Chart

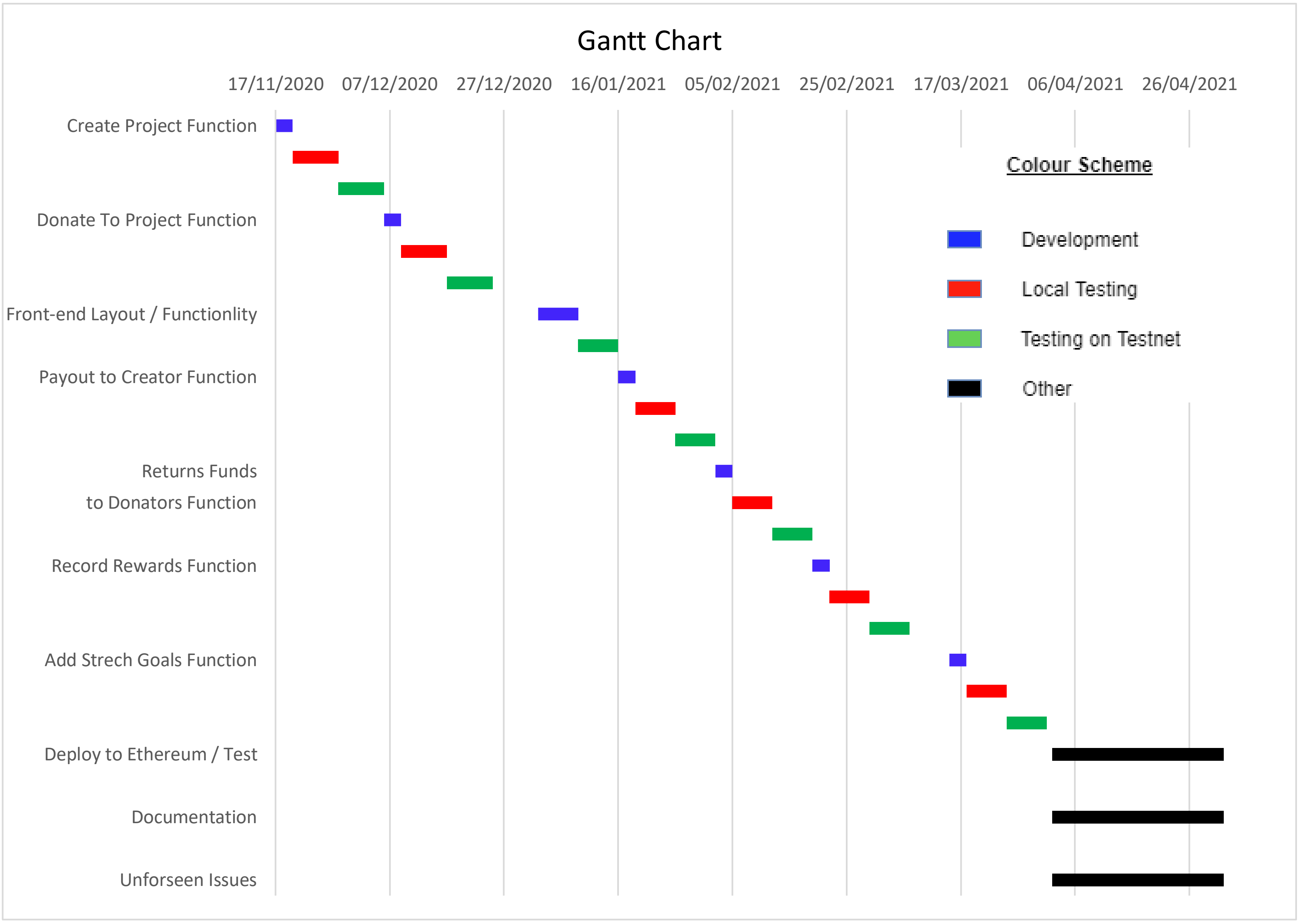


Fig 6.1

7. Appendices

- Information about the Kickstarter website and how they implement a rewards based crowdfunding model: https://www.kickstarter.com/about

- Ethereum documentation: https://docs.ethhub.io/

- The Solidity programming language documentation: https://docs.soliditylang.org/en/v0.7.4/

- Go Ethereum (geth) documentation: https://geth.ethereum.org/docs/

- Web3.js API documentation: https://web3js.readthedocs.io/en/v1.3.0/