

CODE SHARING APPLICATION ON ETHEREUM

Project report submitted in partial fulfillment of the requirement for
the degree of Bachelor of Technology

in

Computer Science and Engineering/Information Technology

By

Sourabh(151358)

Sahil Thakur (151360)

Under the supervision of

Dr.Geetanjali (Assistant Professor-Senior Grade)

to



Department of Computer Science & Engineering and Information
Technology

**Jaypee University of Information Technology Waknaghat,
Solan-173234, Himachal Pradesh**

Candidate's Declaration

We hereby declare that the work presented in this report entitled “ **Code Sharing Application on Ethereum**” in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology in Computer Science and Engineering/Information Technology** submitted in the department of Computer Science & Engineering and Information Technology, Jaypee University of Information Technology Waknaghat is an authentic record of my own work carried out over a period from August 2018 to May 2019 under the supervision of **(Dr. Geetanjali)** (Assistant Professor (CSE)).

The matter embodied in the report has not been submitted for the award of any other degree or diploma.

Sourabh(151358)

Sahil Thakur(151360)

This is to certify that the above statement made by the candidate is true to the best of my knowledge.

Dr. Geetanjali

Assistant Professor

CSE

Dated:

ACKNOWLEDGEMENT

We take this opportunity to express our profound gratitude and deep regards to our guide Dr. Geetanjali (Assistant Professor , CSE) for her exemplary guidance, monitoring and constant encouragement throughout the course of this project. The blessing, help and guidance given by her time to time shall carry us a long way in the journey of life on which we are about to embark.

We are also obliged to staff members of JUIT College, for the valuable information provided by them in their respective fields. We are grateful for their cooperation during the period of our assignment.

Lastly, we thank almighty, our parents and our classmates for their constant encouragement without which this assignment would not have been possible.

Contents

1	Introduction	1
1.1	Introduction to Project	1
1.2	Problem Statement	3
1.3	Objective	4
1.4	Methodology	5
1.5	Oraganization.....	7
2	Literature Survey	8
2.1	Books, White Papers and Publications	8
2.2	Talks, Conferences and Meetups	9
3	System Development	10
3.1	System Architecture.....	10
3.2	Experimental Model	11
3.3	Behind The Scenes.....	15

4 Algorithms	17
4.1 Algorithm 1	17
4.2 Algorithm 2.....	18
4.3 Algorithm 3.....	20
5 Testing	21
5.1 Testing Rails Authorization	22
5.2 Unit Test For Repository and Users	24
5.3 Smart Contract Testing	26
5.4 Things Left For Testing.....	27
6 Issues And Challenges So Far	28
6.1 Scalability in the Ethereum Network.....	30
6.2 Future Changes and Expectations.....	33
References.....	34

List of Figures

1.	The Overview.....	7
2.	Devcon IV	9
3.	The Layers Of The Applications.....	10
4.	Flowchart For CheckOwner.....	17
5.	Flowchart For Create New Repository	19
6.	Flowchart For Upload Files	20
7.	Blockchain Trilemma.....	32
8.	Create Repository Screenshot.....	36
9.	Login into application screenshot.....	37
10.	Private Blockchain.....	38

List of Graphs

1. Number of Users On Github.....4
2. Prices For Ether Over The Last Three Years.....12
3. Scalability In The Ethereum Network.....28

List of Tables

1. Rails Authorization Test.....	23
2. Unit Tests For Repository And Users.....	25

List of Abbreviations

1. PoW – Proof Of Work
2. DApp – Decentralized Application
3. PoS – Proof Of Stake
4. IPFS – Inter Planetary File System
5. ICO – Initial Coin Offering
6. TDD – Test Driven Development
7. UI – User Interface

ABSTRACT

Code in today's world is one of the most important entities a person can possess so it is highly vital to store and share our code in a safe and efficient manner. Now, many services like Github, Gitlab etc. do exist that give us many of the functionalities that we desire but what they lack is absolute security and treating code as an 'asset' rather than treating it as data. To tackle this and treat code as valuable we need something different from the conventional client-server model.

In this project we seek to build a code sharing platform on the Ethereum blockchain that will give us the all-important decentralization that developers have been seeking since long and give our code a safer and more permanent home where it is treated as value instead of just simple data.

Our hope is that in the future a bigger more scalable implementation of our idea will change the code sharing perspective in the world.

CHAPTER 1: INTRODUCTION

1.1 Introduction

The World Wide Web was developed in the 90's and it is basically the center point of our life, our businesses, our transportation and every other utility imaginable to human beings. It was revolutionary no doubt, god-like even but it had its limitations; limitations, which we are ignoring these days as users and organizations are as providers.

The World Wide Web working on the HTTP protocol was what can be termed as the "Internet Of Data". It was created to share information online, information like text, images, videos and information that represent commodities that can be shared with friends, family and even the general public easily and without loss to the one sharing it.

30 years into the future and we are in a very different place technologically, a place where the World Wide Web and HTTP are becoming liabilities on us. The internet we require now is no longer the "Internet Of Data" but the "Internet Of Value". The internet we use right now is used for sharing data, no doubt, but there's so much more being shared on the internet now – money, assets, and even forms of data that could may very well be worth a lot of monetary value.

It is pretty evident to developers and organizations that HTTP and World Wide Web working on the client-server model are just not good enough to suit the demands of the "Internet Of Value". So, why are we still running in the old world? Well, there are two major reasons for this – we fear change as users and most importantly organizations fear change as providers. It isn't that solutions haven't been developed, but that the solutions are not being opted due to this fear. Sure, the solutions might not be totally ready for use in every aspect of the world, but they will never be unless we give them a chance. Its 1990's all over again, people didn't want to go online – they trusted their hard copies, they trusted their one-to-one interaction more.

We now are in exactly the same position as developers, companies, users were in the 90's. Now, it's our turn to embrace the change and work with it so we may again have 30-50 years of pain free technology.

The solutions we are talking about are the solutions we are using in our project to solve a particular field in the exchange of data. Data as we know nowadays is one of the most precious assets people own. What form of data is one that is the most valuable? Well, the data that basically runs all the other data in the world as of now – CODE. Code is what runs the internet right now, code is what runs all the digital world and we all have to agree that our world is just as much digital as it is physical these days. From housewives watching TV at home in the afternoon to scientists in NASA working and controlling the ISS, everyone is working on digitalized equipment that has code behind it. What a shame would it be if this code isn't safe? From a simple 'hello world' program to the entire codebase of Facebook is kept in servers somewhere and that is exactly the thing we are going to tackle now.

So, we have been saying that there are solutions to solve what might seem like a lot of problems related to the "Internet of Value" but just what are these? We haven't even named them so far and there's just a single word for it – Blockchain.

Blockchain is the change we are failing to embrace and it is exactly the solution to our problem. We'll discuss the how and why of Blockchain in the later sections.

So, what this project aims to accomplish (on a small scale) is to devise a code storing and sharing application on the Ethereum blockchain. Ethereum is a blockchain that allows us to develop decentralized applications (DApps) over it. So, it is more or less a Github clone that would be distributed and decentralized. Let's see in the rest of the chapter what problems would be solved by this and how these problems will actually be solved.

1.2 Problem Statement

Code sharing and storing has held a very important position in the eyes of developers and coders ever since coding started itself. Local machines were initially used to store code belonging to one person. But this had one limitation, more than one person could not collaborate easily. Moreover, there was no system that gave the developers ability to keep versions of their code (an important practice in development).

Then came git, a version control system that was easy, efficient and brought a sigh of relief to developers all over the world. Built on top of git was a web application that would change code sharing and storage forever released in 2008 called Github, which was just bought by Microsoft for 8 billion USD. Github was everything that developers wished for all along, it was open source, it was free, it was easy to collaborate over, it was easy for version control and provided tons of other features.

But there was, in fact is one major drawback to Github, it works on the client-server model at the end of the day. What does that mean? That means that somewhere in the world there is a(one or more) server which contains your code. Let us say for an example that your profile's code is stored in a server in Kentucky, USA. What if there was an earthquake in Kentucky and all the servers broke? Well, say goodbye to your code if you didn't have a backup. This is just a fictional natural calamity that might occur. Consider a more probable example, what if a hacker got control of the server and caused all sorts of disruptions in your code?

What is the problem with this client-server model? Well, there's just a single point of truth and storage. Our entire web is client-server right now, and this isn't a very good thing if we want to store value on the internet! If our value is stored at a single point of truth, it is more or less inviting trouble.

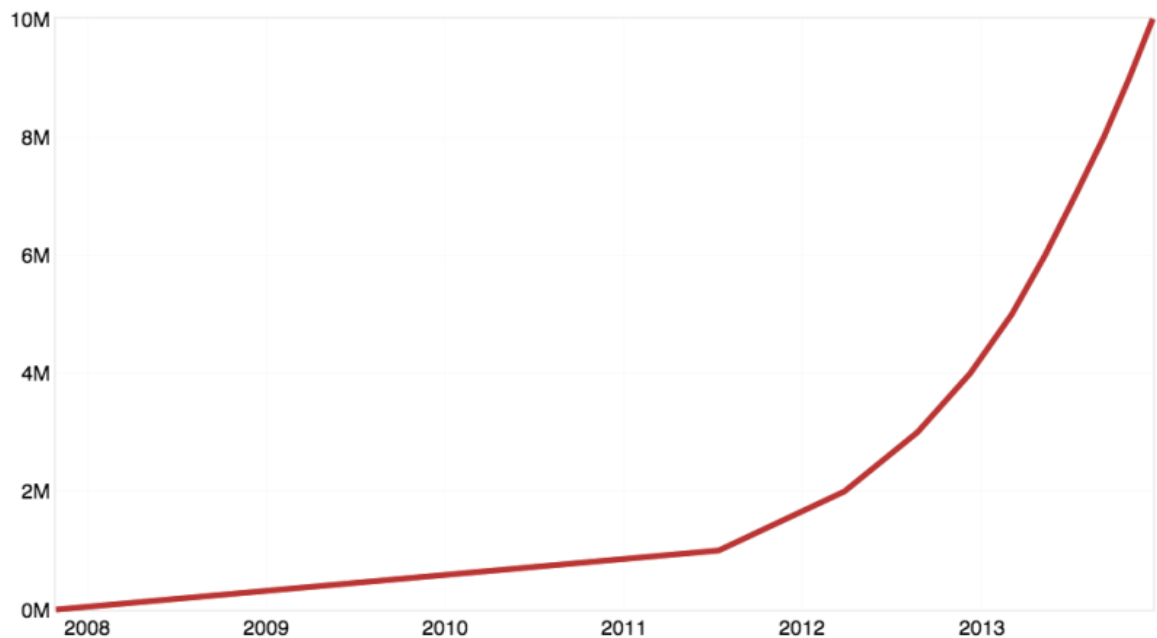
What we want to do is make our code-sharing and storing distributed and secure. The method to do it is using Blockchain technology. Client-server is not the solution to it, it never was but we didn't have the technology required to make something better. Now, we do.

1.3 Objectives

By now, we have laid down certain things we expect from our application and certain things we demand from it. We expect everything that Github is capable of, albeit at a smaller level and we demand three things:-

- i.Distributed
- ii.Trustless
- iii.Secure

Ethereum will let us develop a code sharing web application that will satisfy all of the above demands. Our application will be what is called a DApp. Decentralised applications are peer-to-peer so they don't need any central server (that can be attacked) , they follow consensus rules (to maintain the state of the blockchain) and are secure because once the data goes below a certain no. of blocks, that data is immutable no matter what.



Graph 1.1 – No. of users on Github

There are more than 15 million users on Github alone, so this will be a big problem that we are solving, and more importantly, it is a very precious resource that we are keeping secure – CODE.

1.4 Methodology

So, how will things go about as we said they will? Over the past few years there's been a trend in the technological universe called ICO (Initial Coin Offering) that let's blockchain developers release their coins into the market that can be bought in exchange of fiat currency. These coins can then be exchanged for the services provided by the application. This is replicated in the Ethereum blockchain using something called tokens. Tokens are sold in ICOs and then later can be used for services.

What our application aims at doing is providing tokens in the ICO in exchange of fiat currency. Now, the number of tokens is always capped, so more the demand of tokens, higher the tokens go in value. Our application's tokens would be used in exchange of storing code on the application and adding collaborators to your code. Even though this approach isn't free of cost to the end users, it might very easily be provided freely if the developer feels like open-sourcing the project (which will lead to a consensus clash we will discuss later).

Pieces of Solidity code called Smart Contracts are what maintain peace on the Ethereum blockchain. Smart contracts are what we will use in order to regulate our web application. These smart contracts will decide how many tokens will be used for what operation and which files belong to which user.

We discussed before that HTTP is not suitable for the exchange of value, so in comes IPFS (Inter Planetary File System). This is the protocol that will be used by our application to store and communicate data , i.e, the codes. We will discuss more about how IPFS will do it and how will implement all this in later sections.

At the end of it all, our application will just be a web application that is built using tools like Ruby On Rails and ReactJS with the speciality that it will run on the Ethereum blockchain that will give it benefits unheard of.

1.5 Organization

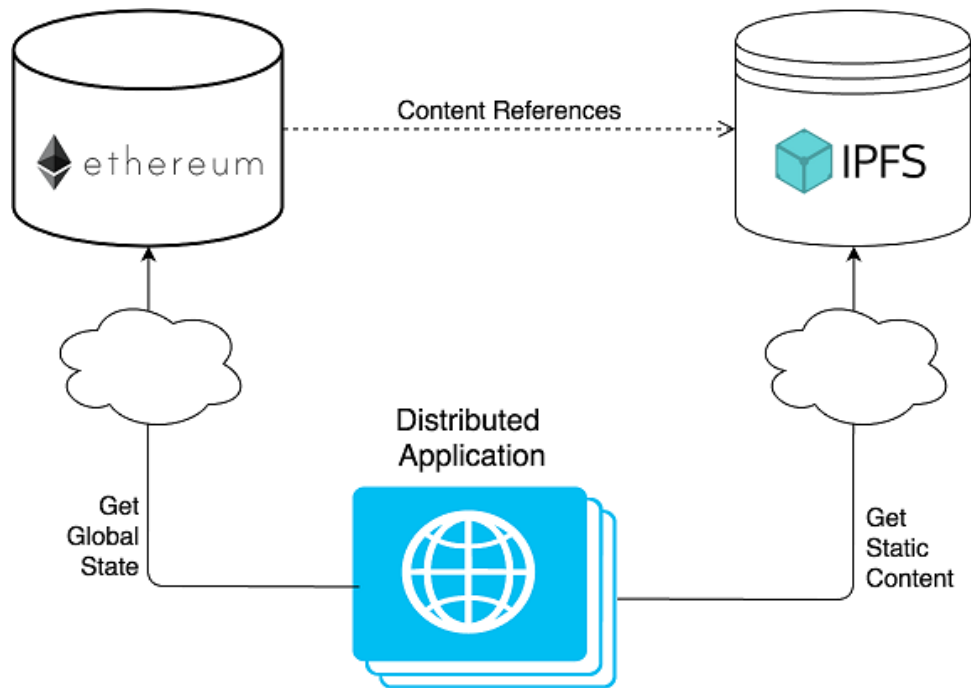


Fig 1.1 The Overview

Our application will be organized into four major different parts – Ethereum solidity smart contracts, Ruby on Rails backend application, ReactJS front-end UI and the IPFS file system.

The Rails backend will provide the functionalities of a code sharing application like Github (interestingly, Github is built on Rails too!), ReactJS will provide users with the interactive front-end to deal with the Web3 providers at one end and the users at the other. The web3 that React will interact with is responsible for talking our smart contracts which will be governing the decentralised logic of our application. IPFS is what will be used as a file system for the application as we cannot rely on a centralised file system in a blockchain application.

Our application state will be handled at all these different levels differently in accordance to the requirement by the tool handling it.

CHAPTER 2: LITERATURE SURVEY

2.1 Books, White Papers and Publications

Decentralised applications are not something thought of in 2015 by Vitalik Buterin, developers have always been on the hunt for achieving decentralisation since the very moment they found the flaws in a centralised system.

Vitalik just brought enough innovation into the Bitcoin blockchain to find a method to host Decentralised applications.

Ethereum White Paper -> <https://github.com/ethereum/wiki/wiki/White-Paper>

Bitcoin White Paper -> <https://bitcoin.org/bitcoin.pdf>

The release of the Bitcoin whitepaper in 2008 was a big event in decentralisation of the world. An unknown person(s) called Satoshi Nakamoto released this paper that seemed to have solve Byzantine's problem of decentralisation. The entire Bitcoin protocol and basically how blockchain technology works as whole is very well explained in the book *Mastering Bitcoin* by Andreas Antonopoulos.

The thing with distributed ledger technology (other name for blockchain) is that it is not a single technology working on its own – its cryptography, its networking, its web application development, its consensus theorems all working together to form DLT.

Two of the books really shedding light on these are:-

- i. Distributed Networks by Qurban A. Menon
- ii. Handbook Of Applied Cryptography by Alfred Menezes

Other than this regular magazines like *NXTR* and *CryptoBiz Magazine* are constant sources of information in the blockchain and crypto world.

The *International Journal Of Blockchain And Cryptocurrency* also exists for authors to send their research papers to and get heard in this space by other technology enthusiasts.

2.2 Talks, Conferences and Meetups

The blockchain developer ecosystem is one of the most buzzing in this age. People are working in decentralised applications, in bitcoin, in mining and all other sort of blockchain related works all over the world.

In order to bring these innovative minds together regular crypto meetups are held all over the world. Few of the most important ones are – *World Blockchain Forum* (London, 2018), *Malta Blockchain Summit* (Malta, 2018), *Ethereum Devcon* (Prague, 2018).

The Ethereum Devcon 2018 was a particularly important even in the blockchain space considering it was where the Ethereum foundation laid down their visions for Ethereum 2.0 set to be released in 2020 with particular focus on scalability and performance.



Fig 1.2 Devcon IV

CHAPTER 3: SYSTEM DEVELOPMENT

3.1 System Architecture

The application will be architected on various levels using various different tools – the bottom layer would be the Ethereum blockchain that will act as the underlying ledger to record everything that is of importance according to the smart contracts, the layer just above it would be the IPFS file system that would allow us to store the code files in a peer-to-peer network. The third layer from the bottom would be the Smart Contracts layer written in Solidity. The next layer would be our backend web application layer written in Ruby. On top would be the ReactJS layer written Javascript for the users to interact with.

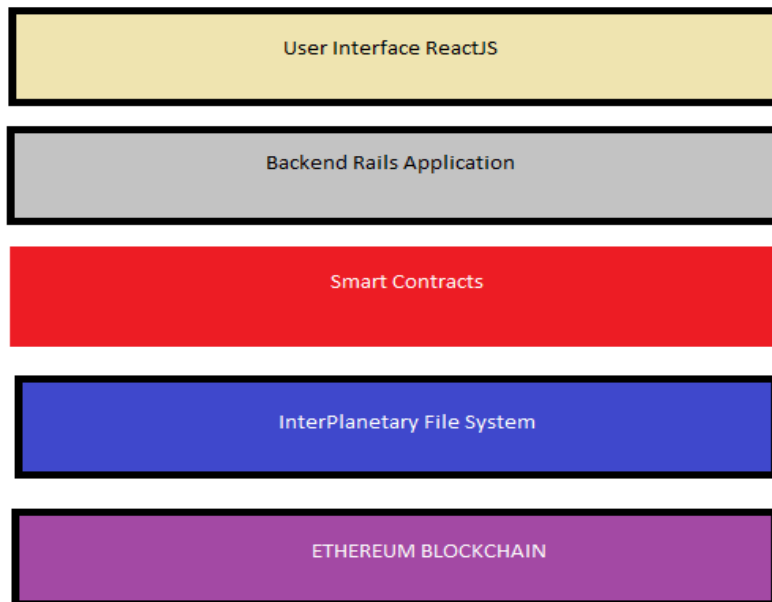


Fig 3.1 The layers of the application

3.2 Experimental Model

Our system is devised on the ICO which will be offering tokens that can be then used in exchange of services provided by the app in the future. Now, to run applications on Ethereum, the user needs to spend something called ‘gas’ that is used to power the computation on the web3 provider that is actually executing the code somewhere. Remember that the code isn’t being executed on a server like in centralised applications, therefore it is necessary to spend some amount of gas (which can be bought with Ether, the currency on the Ethereum blockchain).

So, let us assume our user A has bought some tokens of our application in the ICO, what does he do then? Well, he uses his browser (with Metamask enabled, which is an extension to run blockchain apps) to open our web application. It is noteworthy that we will only talk to the blockchain when we really need to, rest all of the operations are to be carried out by our Rails backend.

After visiting the application, our user logs into the application and visits his profile page. Now, the interesting part starts – as soon as he creates a repository (a data structure) our application will talk to our smart contract using Web3 and start writing information on the blockchain. It is necessary because repositories are what will be written to the blockchain and they are the core building blocks of our application. These repositories are what will be governed by our smart contracts.

Our user creates a repository and our smart contract writes that information on the blockchain that this repository belongs to user A. This is an operation that costs ‘gas’ on the blockchain and the token that we distributed in the ICO for availing the create repository service of the application. That no. of tokens will be deducted from A and given to the person that mines the block containing the transaction with the create repository function call. This is what keeps the miners incentivised for keep on working to mine blocks.

The block with that transaction gets attached to the Ethereum blockchain and thus gets settled. Once there are more than 7 blocks on top of that block (1 block gets mined every 15 secs), that information becomes immutable! Therefore, no one in the world can change the information that that particular repository belongs to User A.

All the features provided by the application like storing your files in a repository, collaborating in someone else's repository and making any other change to your repository will cost gas as well as the tokens of our application.

Also, it is worth noting that read operations will not cost gas or tokens as they are not writing any data on the blockchain and thus can be executed on the user's own local node. Write operations are the one which spend computation power and therefore need to cost something. The Web3 provider is the node where all the computations take place.

Metamask is an extension for Chrome, Firefox etc. which allows you to interact with any Ethereum application using your account. You can configure your web3 provider using Metamask though it is default set to Infura (which provides a node layer for computations). Now, the more the demand for your application, more the value of your tokens in fiat currency and same is the case with Ether, more the demand for decentralised applications, more the cost of Ether.

Ethereum Charts



Graph 3.1 Price of Ether over the last three years

The prices of Ether, which affects the price of gas and also the price of our token is an important thing to consider in decentralised applications. It would directly affect the usage and value of our application.

The main flow is nearly the same for any operation carried out in our application, it's basically spending tokens and gas to get your operations done.

Let's see an operation to upload a code file. User A will open our URL, login to our Rails application, will open one of the repositories that he has already created (won't cost gas as it is a read operation). Now, when he clicks the upload button with the files that is when a smart contract function is triggered. First the function will test whether the msg.sender is the owner of the repository or not, if true, the files will be uploaded to IPFS and their hashes will be written on the blockchain belonging to that particular repository.

In the future when the user tries to access these files, it will be checked whether these files belong to a repository that is owned by that User A. The data in the blockchain is immutable, therefore, there's no chance of your data being interfered with by anyone or any changes being made to your data or profile.

```

1 repository.sol
22 pragma solidity ^0.4.19;
21
20 contract GitProject {
19     mapping(address => uint) numOfRepos;
18
17     mapping(string => address) ownerOf;
16
15     function checkOwnerOf(string _repo) public returns(bool){
14         if (ownerOf[_repo] == msg.sender){
13             return true;
12         }
11         else{
10             return false;
9         }
8     }
7
6     modifier onlyOwner{
5         require(msg.sender == ownerOf[_repo]);
4         _;
3     }
2
1     function uploadFiles(string _repo) public onlyOwner{
23         //Code still to be implemented
1     }
2 }
~
~
~

```

3.3 Behind The Scenes

We've seen how the user will interact with the application and how the things will go on from the user's perspective but let us see what happens behind the scenes when operation is carried out in our application.

The top layer of our application is the ReactJS UI that we expose to the user. The ReactJS UI of our application is talking to our Rails backend using API end points the Rails application has exposed. Also, React is talking to our smart contracts using Web3.

Let's take the example of the user creating a repository. So, User A types the URL into the browser, which triggers a controller action in our Rails backend. Rails provides data in JSON format to our ReactJS front-end which shows it to the user using Javascript. The user clicks on login and logs into his account. Authorization is taken care of by our Rails backend. Any operation that does not require anything to be written onto the blockchain will be taken care of by the Rails backend, shown to the user by ReactJS. The JSON data provided by the Rails application will be processed by the ReactJS front-end and presented as DOM elements using JSX.

Now, when the User A executes any operation that needs the blockchain to be concerned, a different flow runs. Instead of asking the Rails backend for JSON data, the React front-end asks Web3 to talk to our smart contract written in Solidity. The solidity code runs on the assigned Web3 provider, writing (or reading) to the blockchain in the process consuming gas as well as our application's native tokens. The consumption of gas is dependent on the complexity of the operation being executed whereas the consumption of the tokens is dependent on us. Let us assume creating a repository costs 3 tokens. So, after the repository is written on the blockchain and our user is assigned as the owner of that particular repository our smart contract will execute the code that deducts 3 tokens from that user's address.

How does the information like user's address get into the smart contract? The answer is Metamask. Metamask is what contains your address and your balance. Whenever you execute any blockchain operation in the application a transaction gets written! This transaction is signed by your digital signature using Metamask (metamask is where your credentials have been set up). So, the signature on the transaction is where all your information resides. This is

where the smart contracts get information about you. This information is vital because it gets written to the blockchain as the owner of the repository being created.

Later, this information will be required to access the files that you upload onto the application and all the other features in the application that are requiring the blockchain.

After you send your transaction, that transaction along with thousands of other transactions all over the world will be written to a block. Then there will be race between various mining nodes to mine this block, the node that wins this race will get rewarded with transaction fees and ether! All other nodes have to then validate that the information is valid, if not, the block gets rejected, and else it gets written on the main net blockchain of Ethereum which is as good as being engraved on the strongest stone possible. This information is now immutable and can never ever be changed or hacked.

One block gets written onto the blockchain every 15 seconds and after a certain no. of blocks there is no way to change the information written on an older block. Thus, your repository (and later code) becomes as secure as possible. Thus, completing our decentralised operation.

3.4 The Interplanetary File System

Our report has again and again mentioned the fact that HTTP was feasible only for the web of the past. The IPFS approach to file sharing will greatly boost our application and its usage to share code in a decentralized way. HTTP downloads from a single source at a single time, whereas IPFS makes it possible to download from various different sources at the same time. What this approach gives us is gives 60% saving on bandwidth costs. Let us take a step by step look into how IPFS works :-

- a. Each file to be stored via IPFS and all the blocks inside it are given a cryptographic hash as a unique identifier.
- b. IPFS removes duplication of the file across the network.
- c. Each node stores only the content that it has interest in, and some information that will help in figuring out who is storing what.
- d. IPNS (Inter Planetary Naming System) is used to name the file in a human readable form.
- e. When looking for files on the system, you're asking the system to give you content behind a unique hash.

CHAPTER 4: ALGORITHMS

Algorithm 1: Checking Repository's Owner

User.clicks (repository's link)

-> Web3 calls smart contract's CheckOwner function

-> CheckOwner(_uint address, Repository repo)

->for address and repo

-> check in AddressRepo mapping if the value returned true

-> If true, give user the access to the repo

-> If false, give an error message

->end

->end

->end

The thing with this algorithm is that it won't cost any gas or tokens as this is a read operation.

Other operations that require writing to the blockchain are much more risky as we'll see later.

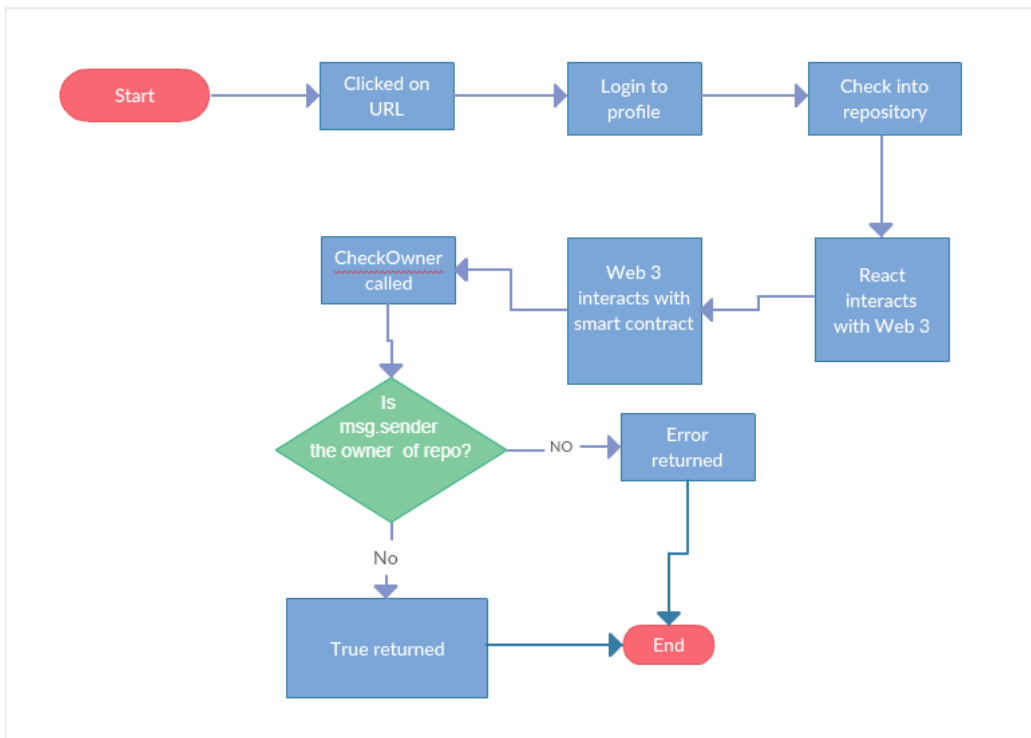


Fig. 4.1 Flowchart for checkOwner

Algorithm 2: Creating a new repository

User.clicks (create new repository)

-> Web3 calls smart contract's newRepo function

-> newRepo(uint _address , String repo)

-> for address and string repo

-> check if repo of same name already exists

-> if yes, ask user to give another name

-> if no,

-> create a new repository, write it on the blockchain

-> assign the repository's owner to msg.sender

-> deduct the specific no. of tokens from msg.sender

-> end

->end

->end

This algorithm allows user to create a new repository and register it on the blockchain, this is a write operation and thus will consume gas as well as tokens.

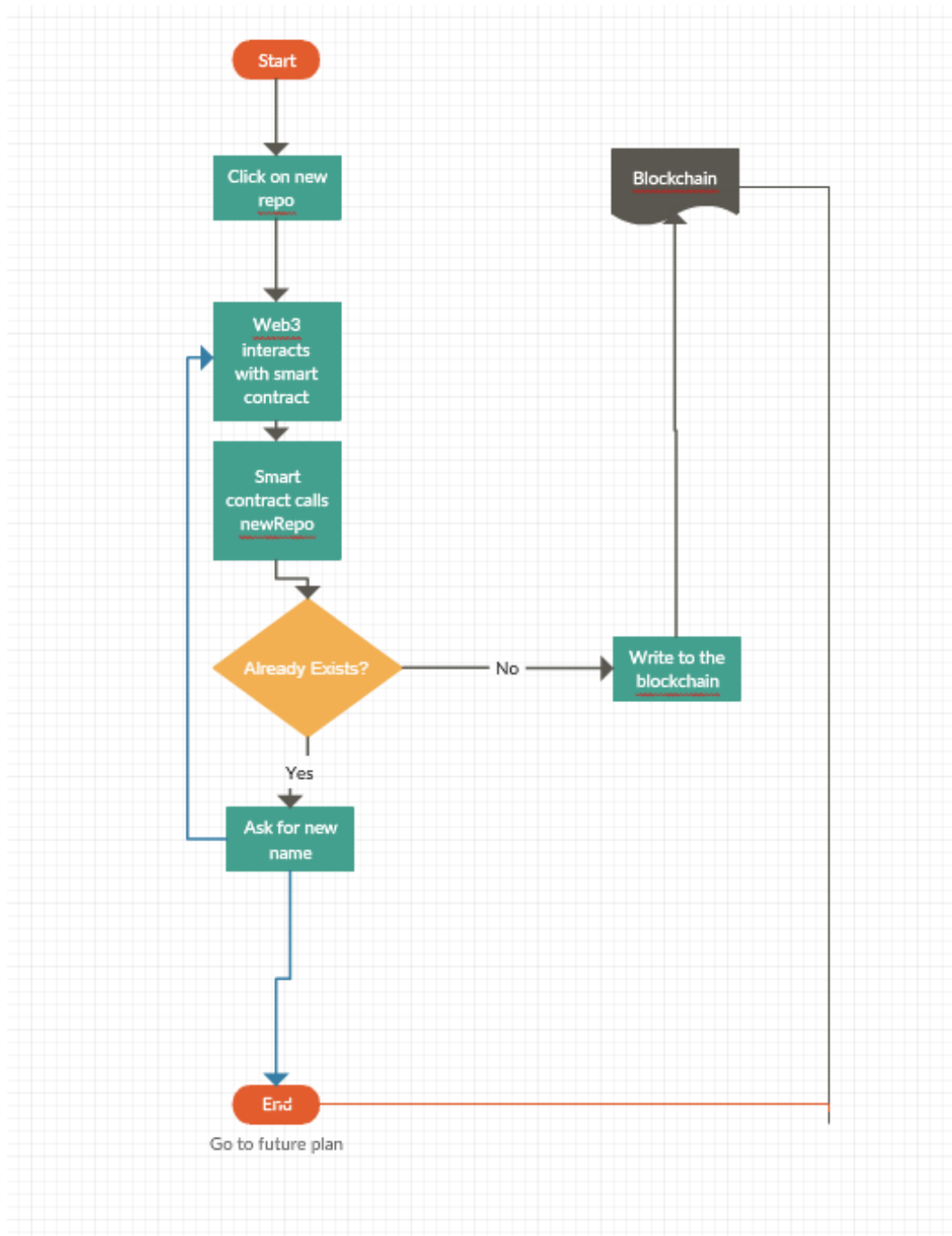


Fig 4.2 Flowchart for create new repository

Algorithm 3: Upload new files

User.clicks(repository repo)

-> Web3 calls checkOwner function

-> if true, allow access into the repo

-> if false, block access into the repo

->User.clicks(upload files)

-> Files get uploaded into the IPFS

-> The blockchain writes these files as belonging to the repo

->end

->end

->end

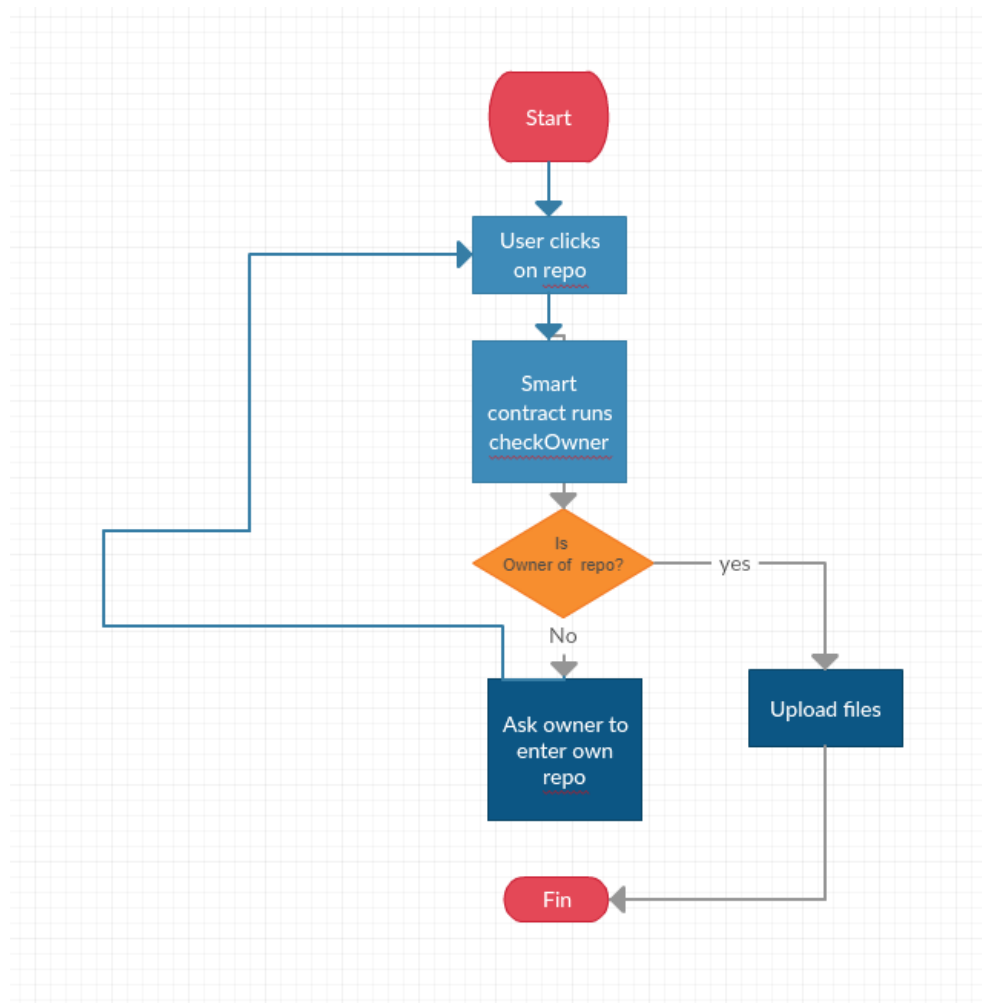


Fig 4.3 Flowchart for upload files

CHAPTER 5: TESTING

We need to test two different parts of our application – the Rails backend and the smart contract solidity code. The testing of the solidity code is of utmost importance as it literally costs money to execute that code.

Reasons to test your Solidity smart contract code:-

1. It costs gas to run functions on Ethereum blockchain and thus any wrongly written function is costing the user money in the form of gas.
2. The smart contract code is immutable, i.e, we cannot change it as we please like we do in a production deployed web application.
3. Once wrong code gets written on the blockchain, it can only be changed if erase it off the blockchain and then write new code in its place.

Reasons to test your Rails back-end code:-

1. Authorization and authentication is of utmost importance in any web application.
2. Other backend services are totally dependent on the Rails code.
3. Our Rails back-end is what provides our UI with JSON data.

Now, we've seen why it is important to test our code on both the levels. We accomplish these tasks using two different tools – Rspec for our rails code and Mocha for our Solidity code. These are testing frameworks that are specifically developed for Ruby and Javascript code (Solidity can be tested like JS as well). We will provide our application with mock data and mock simulations using FactoryBot and Faker which will use Fake data as fixtures and check the working of the application depending on certain assertions that we will write in RSpec and Chai for Rails and Solidity respectively.

5.1 Testing Rails Authorization

The Rails authorization system will be based on JSON JWT tokens that allow for services and features to be accessed only if the current user is using his own correct token. To test the authorization system we create Fake data of users using Faker and initialize a User factory with this fake data. Factories allow us to make fake models of objects in the application just for testing purposes. Then, we will follow the entire procedure of signing up and logging in using correct password for the retrieval of current token using RSpec request tests.

```
1  require 'rails_helper'
2
3  RSpec.describe 'Authentication', type: :request do
4
5    describe 'POST /auth/login' do
6      let!(:user) { create(:user) }
7
8      let(:headers) { valid_headers.except('Authorization')}
9
10     let(:valid_credentials) do
11       {
12         email: user.email,
13         password: user.password
14       }.to_json
15     end
16
17     let(:invalid_credentials) do
18       {
19         email: Faker::Internet.email,
20         password: Faker::Internet.password
21       }.to_json
22     end
23
24     context 'When request is valid' do
25       before { post '/auth/login', params: valid_credentials, headers: headers}
26
27       it 'returns an authentication token' do
28         expect(json['auth_token']).not_to be_nil
29       end
30     end
31
32     context 'when request is invalid' do
33       before { post '/auth/login', params: invalid_credentials, headers: headers}
34
35       it 'returns a failure message' do
36         expect(json['message']).to match(/Invalid credentials/)
37       end
38     end
39   end
40 end
```

USER	TOKEN	RESPONSE
A	Token.belongs(A)	Access
A	Token.belongs(B)	No-access
A	Token.timeout(A)	No-access
A	Token.invalid	No-access

Table 5.1 User Authentication

5.2 Unit Tests for Repository and Users

Other than the complete flow of user's authorization , it is very important to test each model of the application using Unit tests to confirm each of the individual models are working as expected. We write Model tests for User model and Respository model where we test there trivial and basic behaviour using RSpec.

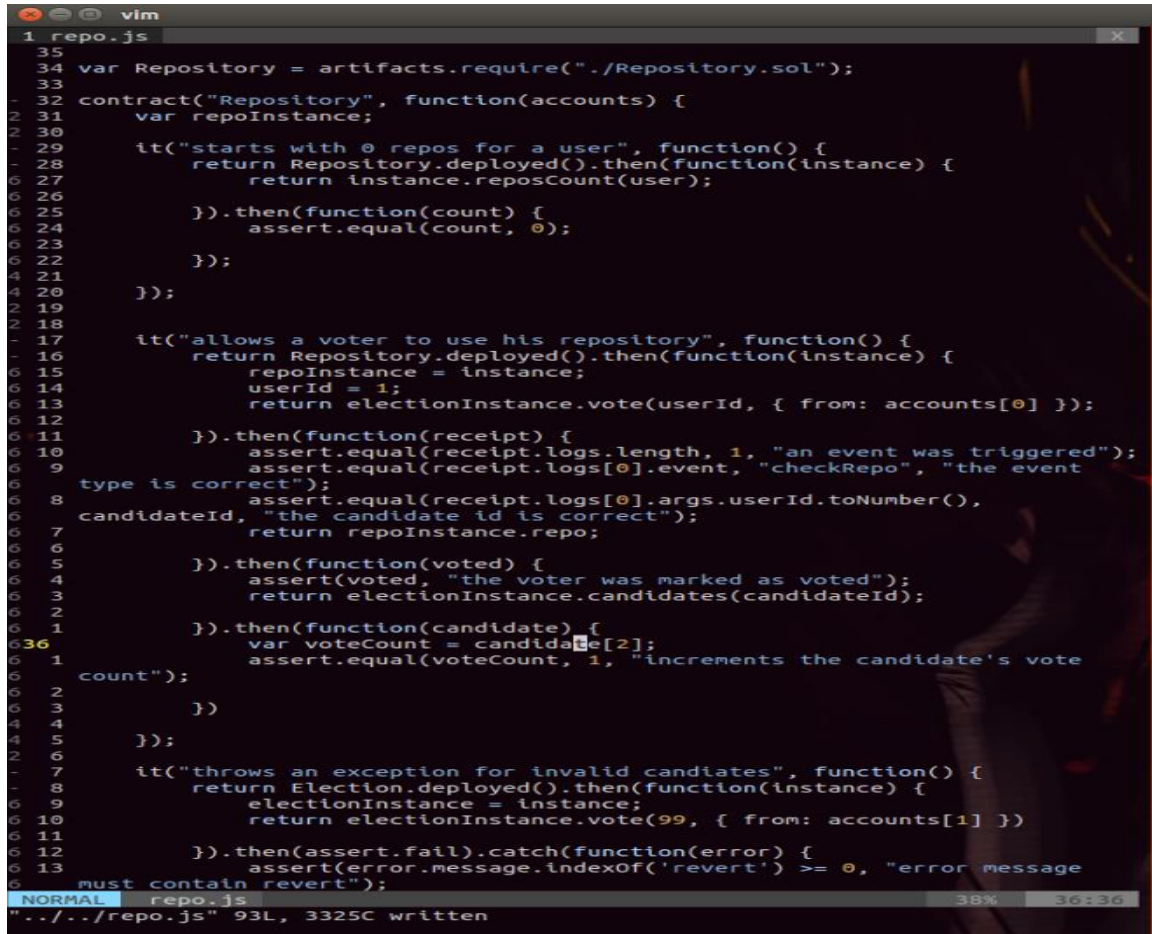
```
1 require 'rails_helper'
2
3 RSpec.describe 'Articles API', type: :request do
4   let(:user) { create(:user)}
5   let!(:category) { create(:category)}
6   let!(:repos) { create_list(:repo, 10, category_id: category.id) }
7   let(:repo_id) { articles.first.id }
8   let(:headers) { valid_headers }
9
10
11   describe 'GET /repos' do
12     before { get '/repos', params: {}, headers: headers}
13
14     it 'returns repos' do
15       expect(json).not_to be_empty
16       expect(json.size).to eq(10)
17     end
18
19     it 'returns status code 200' do
20       expect(response).to have_http_status(200)
21     end
22   end
23
24
25   describe 'GET /repos/:id' do
26     before { get "/repos/#{repo}", params: {}, headers: headers}
27
28     context 'when the repo exists' do
29       it 'returns the repo' do
30         expect(json).to_not be_empty
31         expect(json['id']).to eq(article_id)
32       end
33
34       it 'returns status code 200' do
35         expect(response).to have_http_status(200)
36       end
37     end
38   end
39 end
```

TABLE

Repository Name	Repository Owner	Response
“ ”	“ ”	Error
“ ”	A	Error
Rails Project	“ ”	Error
Rails Project	A	Accepted

5.3 Smart Contract Testing

As we discussed earlier, testing smart contracts is of utmost importance as their code actually costs real money to be run, so , we need to make sure everything runs as expected and using as little gas as possible.



```
1 repo.js
35 var Repository = artifacts.require("./Repository.sol");
33
32 contract("Repository", function(accounts) {
31   var repoInstance;
29   it("starts with 0 repos for a user", function() {
28     return Repository.deployed().then(function(instance) {
27       return instance.reposCount(user);
26     }).then(function(count) {
25       assert.equal(count, 0);
24     });
23   });
22 });
21
20 });
19
18
17   it("allows a voter to use his repository", function() {
16     return Repository.deployed().then(function(instance) {
15       repoInstance = instance;
14       userId = 1;
13       return electionInstance.vote(userId, { from: accounts[0] });
12     }).then(function(receipt) {
11       assert.equal(receipt.logs.length, 1, "an event was triggered");
10       assert.equal(receipt.logs[0].event, "checkRepo", "the event
9 type is correct");
8       assert.equal(receipt.logs[0].args.userId.toNumber(),
7 candidateId, "the candidate id is correct");
6       return repoInstance.repo;
5     }).then(function(voted) {
4       assert(voted, "the voter was marked as voted");
3       return electionInstance.candidates(candidateId);
2     }).then(function(candidate) {
1       var voteCount = candidate[2];
36       assert.equal(voteCount, 1, "increments the candidate's vote
1 count");
2     });
3   });
4 });
5
6
7   it("throws an exception for invalid candidates", function() {
8     return Election.deployed().then(function(instance) {
9       electionInstance = instance;
10      return electionInstance.vote(99, { from: accounts[1] });
11    }).then(assert.fail).catch(function(error) {
12      assert(error.message.indexOf('revert') >= 0, "error message
13 must contain revert");
14    });
15  });
16  });
17  });
18  });
19  });
20  });
21  });
22  });
23  });
24  });
25  });
26  });
27  });
28  });
29  });
30  });
31  });
32  });
33  });
34  });
35  });
```

NORMAL repo.js 38% 36:36
"../../repo.js" 93L, 3325C written

5.4 Things Left For Testing

Due to no connection between our Rails back-end and our smart contracts till now, no form of integration testing has been carried out.

Individual parts of the application and individual parts of those individual parts have been tested independently but they have still not been combined into a single flow of behaviour. To implement integration testing, we first need to completely build the application from the start to the end so that we can test the behaviour and connection of each layer with the layer next to it.

The integration testing will be carried out using the Mocha test framework because it would allow us to test the UI provided by ReactJS as well as the core functionality provided by the Smart Contracts.

CHAPTER 6: RESULTS AND PERFORMANCE ANALYSIS

The project has been working well when the blockchain is run on our own system with no connection to the outer world. The way we are analyzing the performance of the project is by testing it in two ways – firstly we are using a local blockchain using a software called Ganache; what this does is create a local blockchain on our own system with fake users (all of whom are controlled by your Ganache software itself) . The second way we are testing the application is much closer to the real world implementations of a decentralized application, in this second way we are using a test network called Rinkeby.

As to test the application in the real world Ethereum blockchain, we will require real Ether to deploy our contract and real Ether costs real money so we were not able to check the performance of the application on Ethereum. The closest we could get was by using the Rinkeby testnet that operates on fake Ether that can be bought using rinkeby-faucets .

The two major points to check in a decentralized application are the performance and costs. As every operation on a DApp takes gas to operate, it is vital that the operations take as less ‘gas’ and thus as less money as possible. For better security, you need to pay a better price and thus gas heavy operations tend to be more secure even though that is not a compulsion. The other aspect of performance analysis is also very important. As our DApp will be deployed on the Ethereum blockchain, it will follow a proof-of-work consensus algorithm. What this means from a performance point of view is that the scalability of the app goes down and response times increase. The average block time on the ethereum blockchain is of 10 mins. What this means is any operation will take 10 mins to be permanently recorded on the blockchain . Even though using Solidity specific programming concepts of ‘Events’ can give us the ability to change the UI on the front end on a particular event (just like event handlers in JS), the real transaction does take 10 mins to save onto the Ethereum blockchain.

Things are very very different on the Ganache local blockchain where every action is spontaneous , and on the Rinkeby testnet where the average block times vary from time to time.



Figure 6.1 Screenshot of Rinkeby’s performance

The information provided by the Rinkeby test network on their official site (<https://www.rinkeby.io/#stats>) shows the average block time of around 15 seconds which is very good considering the time taken on the original network.

The best way to check our application is by mixing up the performance analysis. As the gas costs depend on the code, which runs in the same way on Ganache as well as Rinkeby and the real Ethereum network, it is good to check Ganache for how much ether (thus gas) is being spent on every transaction that is sent from a particular account when we are surfing through and using the application. It is important to note and remember that gas is costed only on transactions that write to the blockchain, therefore read-only operations cost no gas.

To check the scalability and performance however makes us test the working application on the Rinkeby test network and then later opening the Rinkeby test network’s site and tracking your transaction and your particular block on the Rinkeby blockchain.

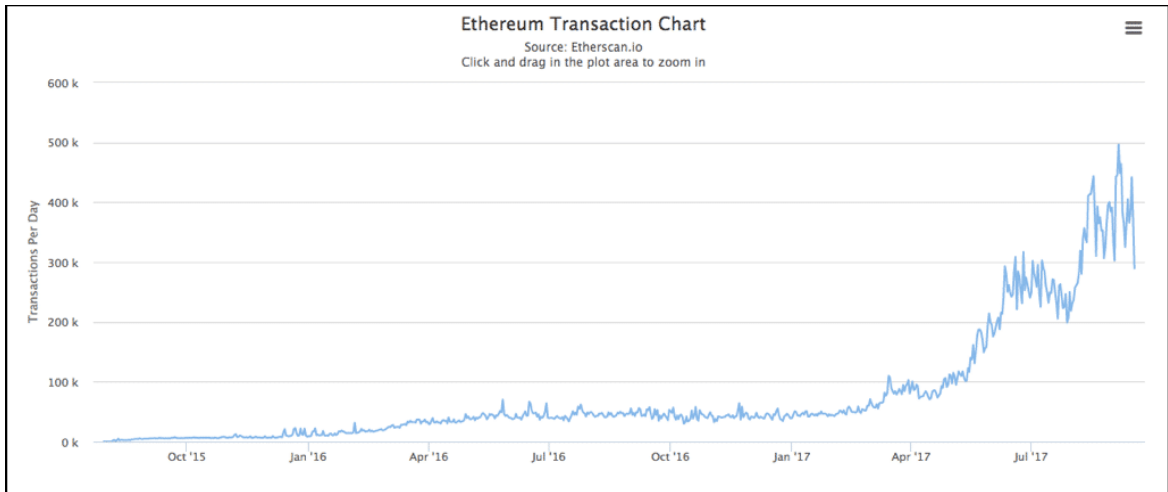
Rinkeby utilizes the Power Of Authority (PoA) accord, explicitly the Clique PoA agreement. The convention makes Rinkeby insusceptible to spam assaults simply like Kovan. Not at all like Kovan, be that as it may, Rinkeby takes a shot at the Geth hub. Additionally, clients can't mine Ether on the Rinkeby, however should demand it from a fixture.

Previously known as the TestRPC, Ganache CLI is a quick and adaptable blockchain emulator for Ethereum. It is both a work area application and an order line apparatus, and is accessible on various major working frameworks, for example, Mac, Windows, and Linux. On the off chance that you are keen on utilizing this test system, you can introduce it by means of NPM, a bundle administrator for Node.js modules.

Ganache CLI has some eminent qualities:

- Users can right away mine their exchanges.
- Transactions are free.
- It needn't bother with a spigot or mining since clients can reuse or reset their records in a flash with a fixed measure of Ether.
- Users can change the gas cost and mining speed.
- It has a GUI where clients can screen their test chain occasions.

CHAPTER 7: CONCLUSIONS



Graph 7.1 Scalability in the Ethereum Network

The biggest problem faced by not only us but anyone who uses Ethereum as their blockchain of choice is that of scalability. Ethereum uses the PoW consensus algorithm which needs majority of nodes to agree on the state of the blockchain before proceeding ahead. What this does is makes the DApps running on Ethereum very slow when trying to change some sort of state.

Decentralisation and Scalability do not often run hand to hand but with the 'Casper' version of Ethereum's consensus algorithm being released somewhere in 2019 or 2020 which will use a partial Proof Of Stake (PoS) consensus algorithm we can expect a major boost in the scalability performances of Ethereum. Other blockchains already running the PoS algorithm like EOS are giving much better performance as of now.

Despite EOS giving much better scalability than Ethereum this very moment, we chose to use Ethereum because it is more decentralised which is the most important aspect of DApps. Scalability is an issue that can be resolved in the future, but decentralisation is what we strive for and that's why Ethereum was chosen over EOS.

Another issue being faced in the development of this application is connecting the Rails backend to the smart contract backend. Till now, most DApps that have been created on Ethereum relied on the smart contracts for their entire code and thus had to interact only with Web3 for the user interaction concerning their application but what we are trying to achieve is to improve the scalability by using the smart contract for only the most important piece of our application that it needs to care of. Rest of the operations are to be taken care of by the Rails backend.

Establishing the connection between 3 major layers would be a task hard to accomplish and TDD would be a much better approach to implement this as being cautious and safe beforehand would be the better mindset than trying to brute force our way into hitting a viable solution.

The Blockchain Trilemma

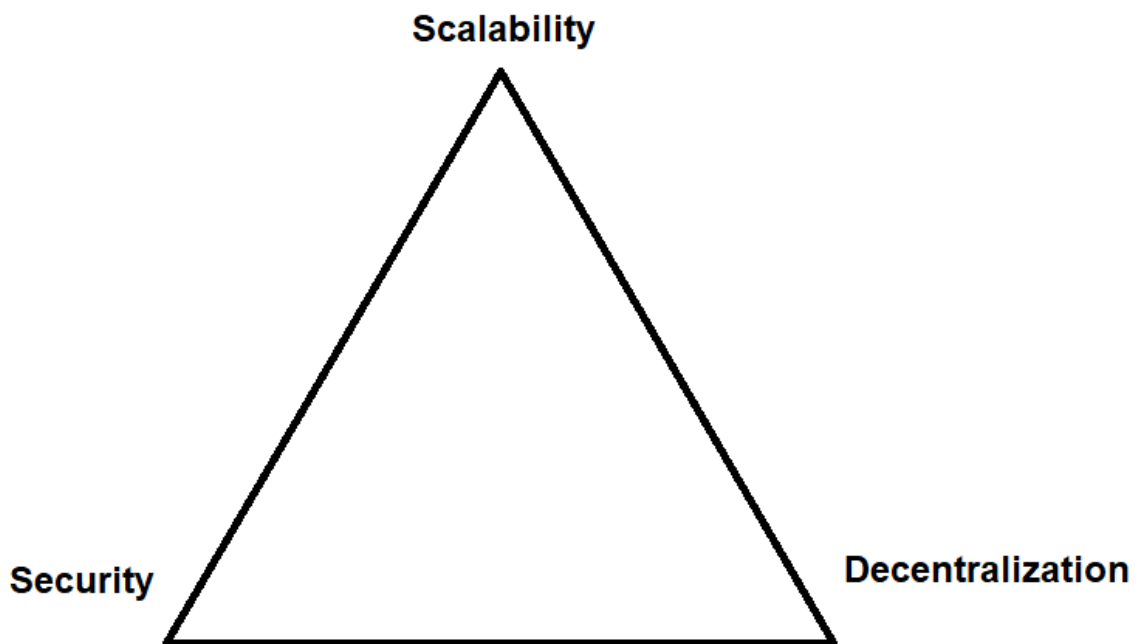


Figure 7.2 The Blockchain Trillema

By far most of dApps depend on a hidden convention. This has generally been Ethereum however new players, for example, EOS, Zilliqa, Stellar are asserting increasingly good conditions for future dApp organization. Regardless of what convention a venture chooses to construct their dApp on there are innate confinements with the innovation of today. A standout amongst the most widely recognized issues is the capacity for a convention to deal with an a lot of clients. Clog on occasion of high utilization can bring the whole system (and any dApps expand over it) to a granulating stop.

This is not to say that all hope is lost, not by any means. There are many protocols out there making big claims (which they may be able to back up). Even Ethereum (considered the premier smart contract protocol) has significant scaling solutions on the horizon. The problem they have is trying to solve the 'blockchain trilemma', that is the competing goals of being secure, decentralised *and* scalable. Most protocols only achieve two of these three goals.

As I would like to think it is too soon to pass judgment on the execution or future accomplishment of dApps. Indeed, many have not satisfied a portion of the promotion, at the same time, is it sensible to figure they would thump on the entryways of Uber, AirBnb and UpWork inside one year of their ICO wrapping up? There are numerous inside and outside variables that should be settled before they can have a genuine shot at taking on the huge folks.

As a side note, usually amusing to see individuals upholding the majority of the astonishing tech behind another convention and how great it will be for the applications based over it, at that point, concurrently state that dApps are nevertheless a hallucination and will no doubt never work. On the off chance that the last is valid, why waste time with the previous?

Boundaries to section, instability, informing, innovation confinements and poor structure are on the whole contributing variables; however there are others.

Be that as it may, trust isn't lost. Numerous dApp ventures have taken in these exercises and are angrily working at bright approaches to conquer them. dApps will arrive yet perhaps we simply need to give them somewhat more time?

6.2 Future Changes And Expectations

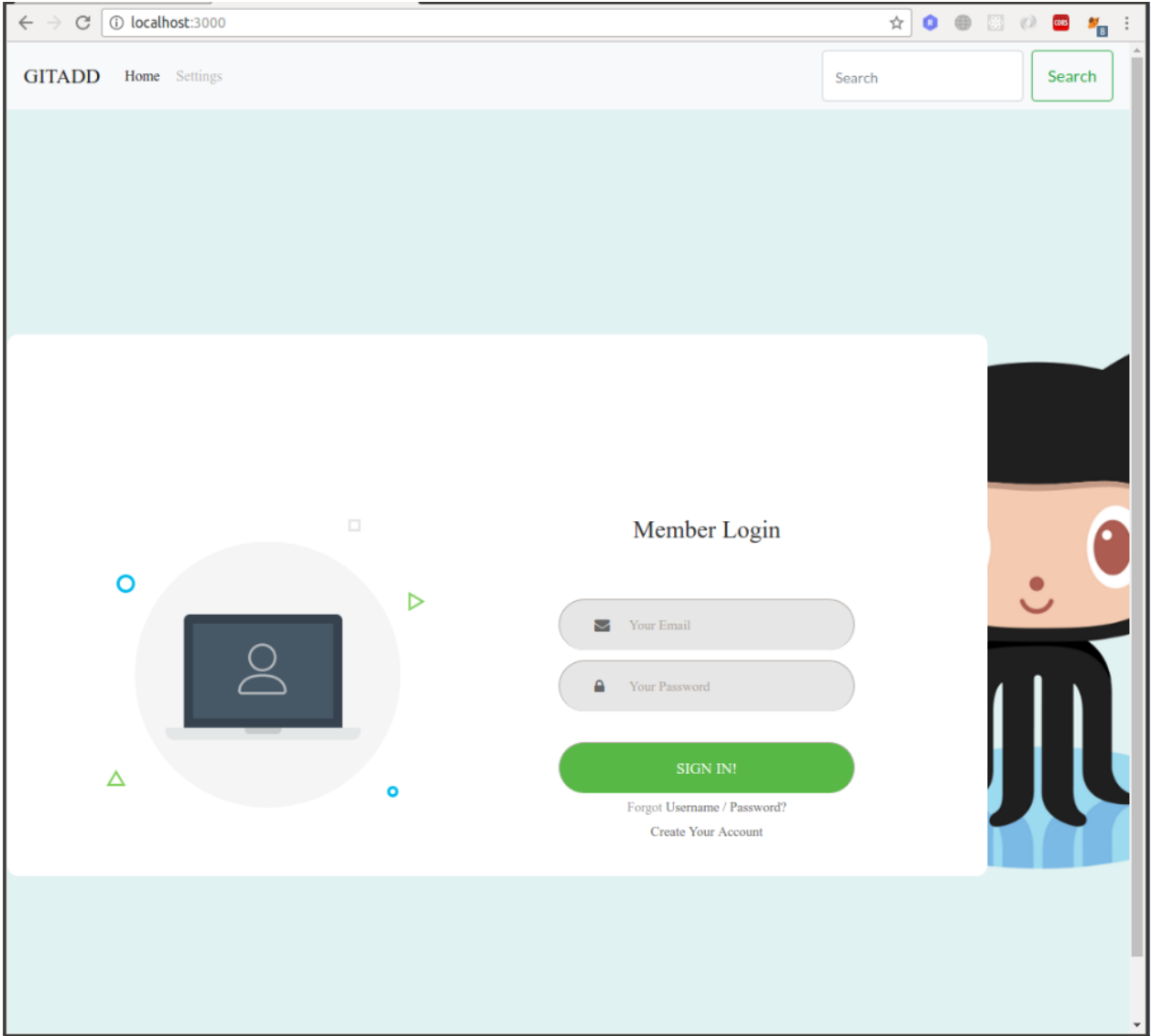
The idea is to follow Test Driven Development to establish a connection between the 3 layers of our application keeping scalability and security at the most importance. Decentralisation is already being achieved on Ethereum so that is not much of an issue. Even if the scalability of the project is not very good, it can be expected to improve many folds when Ethereum 2.0 gets released.

Decentralised applications are here and are here to stay so it important for us to develop them, use them and appreciate the features they provide even in the early stages as 5 years from now the Internet is going to be decentralised and all the shortcomings of the current blockchain technology will be removed and a thing of the past.

Working Screenshots

The screenshot shows a web browser window with the address bar displaying `localhost:3000/users/7/repositories/new`. The browser's navigation bar includes the text "GITADD" and a menu with "Home", "Repositories", and "Settings". A search bar with a "Search" button is located in the top right corner. The main content area has a light gray background and features the heading "NEW REPOSITORY FORM" in large, bold, black letters. Below the heading is a form with four sections: a text input field labeled "NAME", a text input field labeled "LANGUAGE", a larger text input field labeled "DESCRIPTION", and a file upload section containing a "Choose file" button, the text "No file chosen", and a "CREATE" button.

Screenshot 1: New Repository Creation



Screenshot 2: Login into the application

Ganache

ACCOUNTS BLOCKS TRANSACTIONS LOGS

SEARCH FOR BLOCK NUMBERS OR TX HASHES

CURRENT BLOCK: 0 | GAS PRICE: 2000000000 | GAS LIMIT: 6721975 | NETWORK ID: 5777 | RPC SERVER: HTTP://127.0.0.1:7545 | MINING STATUS: AUTOMINING

MNEMONIC ? faculty brush name debris bulk sea mushroom resemble reform inner popular divert **HD PATH** m/44'/60'/0'/0/account_index

ADDRESS	BALANCE	TX COUNT	INDEX	
0xB3805c5Bf4dac1678cDa93CbD3781B01dca89a0A	100.00 ETH	0	0	
0x3fE2D0F746DcF3CbA253ce354c0468C45598e62B	100.00 ETH	0	1	
0x430E21b528743D41F1f9963A50866ac11A8e00AD	100.00 ETH	0	2	
0xbF902840464aD3EE29BB5269494Cbc9e83D7C199	100.00 ETH	0	3	
0xe89F32eF733df6F02015634Eb6ff8b3Ab87C1700	100.00 ETH	0	4	
0xA4db4659A3dA59F35672938467F8aa5E030393a3	100.00 ETH	0	5	
0x284dA0717e925df5af33AF07137458820d6595a0	100.00 ETH	0	6	
0x75A770f2F7C027e6A4F56fca500A4ae5540007C	100.00 ETH	0	7	

Screenshot 3: The private blockchain in operation

References

[1] (Online Resource) Ethereum WhitePaper (Available on:

<https://github.com/ethereum/wiki/wiki/White-Paper>).

[2] (Online Resource) <http://jsonlint.com/>

[3] (Online Resource). <https://github.com/bitcoinbook/bitcoinbook>

[4] <https://www.rinkeby.io/#stats> – Rinkby testnet

[5] (Online Resource) <https://metamask.io/> - Metamask

[6] <https://www.railstutorial.org/book>

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT

PLAGIARISM VERIFICATION REPORT

Date: 10/05/2019

Type of Document (Tick): PhD Thesis M.Tech Dissertation/ Report B.Tech Project Report Paper

Name: Sahil Thakur, Saurebh Department: C.S.E. Enrolment No 151358, 151360

Contact No. 9629010342 E-mail. sirradhena@saurebh@gmail.com

Name of the Supervisor: Dr. Geethanjali

Title of the Thesis/Dissertation/Project Report/Paper (In Capital letters): _____


CODE SHARING WEBAPP ON ETHEREUM

UNDERTAKING

I undertake that I am aware of the plagiarism related norms/ regulations, if I found guilty of any plagiarism and copyright violations in the above thesis/report even after award of degree, the University reserves the rights to withdraw/revoke my degree/report. Kindly allow me to avail Plagiarism verification report for the document mentioned above.

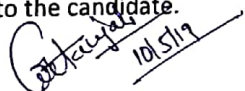
Complete Thesis/Report Pages Detail:

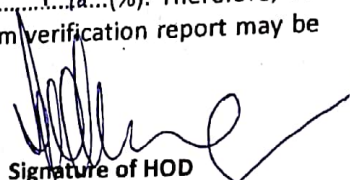
- Total No. of Pages = 50
- Total No. of Preliminary pages = 8
- Total No. of pages accommodate bibliography/references = 1


(Signature of Student)

FOR DEPARTMENT USE

We have checked the thesis/report as per norms and found Similarity Index at 0% (0%). Therefore, we are forwarding the complete thesis/report for final plagiarism check. The plagiarism verification report may be handed over to the candidate.


(Signature of Guide/Supervisor)



Signature of HOD

FOR LRC USE

The above document was scanned for plagiarism check. The outcome of the same is reported below:

Copy Received on	Excluded	Similarity Index (%)	Generated Plagiarism Report Details (Title, Abstract & Chapters)	
			Word Counts	Character Counts
<u>10/05/2019</u>	<ul style="list-style-type: none"> • All Preliminary Pages • Bibliography/Images/Quotes • 14 Words String 20 	<u>0%</u>	6,600	32,086
<u>Report Generated on</u>			Total Pages Scanned	39
<u>10/05/2019</u>		Submission ID	1128208367	
		File Size	1.53M	

Checked by Sahil
Name & Signature Ashok


Librarian 10.05.2019

Please send your complete thesis/report in (PDF) with Title Page, Abstract and Chapters in (Word File) through the supervisor at plagcheck.juit@gmail.com