

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI



PROJECT REPORT

ON

Dexter's Blockchain

MADE BY

GROUP G07

NANE	ID
ABHISHEK KHURANA	2018A3PS0621H
AKANKSHA VINOD HUBLIKAR	2018AAPS0317H
HARSHIT BANSAL	2018B5A70601H

Prepared in partial fulfillment of BITS F452

Blockchain Technology

(September 2021)

TABLE OF CONTENTS

SR No.	Content	Page No.
1.	Introduction & Acknowledgements	2
2.	About the Project	2
3.	Installation and Running the Project	3
4.	Languages & Frameworks Used	3
5.	Operating Environment	3
6.	Virtual Tour of the Project	4
7.	Logic Behind the Working: a. Encryption b. Block c. Blockchain	4
8.	Conclusions	7
9.	References	7

INTRODUCTION & ACKNOWLEDGEMENTS

The following project report entails all aspects of our blockchain application, Dexter's blockchain, covering the four key functionalities:

1. Dexter has information regarding all the available blocks.
2. None of Dexter's friends should be able to edit the added transactions.
3. Timestamp of each transaction is readily available.
4. Dexter should have all the information regarding the completed transactions.

We want to thank Prof G Geetha Kumari, the IC of the course Blockchain Technology (BITS F452), and Harshwardhan Singh for providing us an opportunity to work on this project. We would also like to thank them for sharing their expertise on the skill-set required to fruition this project and their constant efforts in directing us towards the successful completion of the same.

ABOUT THE PROJECT

Project Purpose: Dexter's blockchain is an application that will allow the owner of a business, a coffee shop, in this case, to keep track of all transactions occurring at their shop. This would result in making their business more secure, leading to lesser losses and a more holistic view of how and where the revenue is obtained and spent. This tackles the central issue of other customers removing or editing their transaction logs, leading to significant losses. This project not only helps Dexter to avoid losses but also helps him have a virtual record of all transactions, resulting in a smoother scaling of business and a more efficient working mechanism.

This application aims to make the experience for the user as seamless and user-friendly as possible such that even a user with limited knowledge of blockchain can reap its benefits with ease by securing their business and accessing the various functionalities without difficulty.

We worked as a team on various aspects of our project, including the front-end and backend of our assignment, which entailed creating a chain of blocks, handling transactions, and encryption at different stages.

INSTALLATION AND RUNNING THE PROJECT

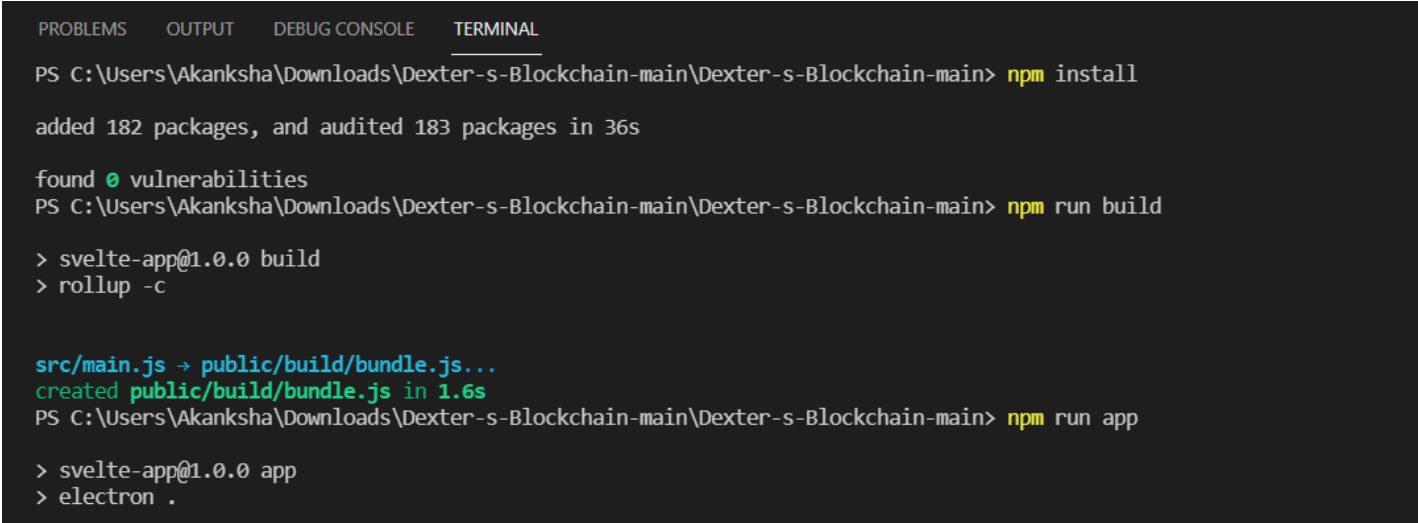
Step 1. open project folder in VScode

Step 2. Type the following commands in the terminal

npm install

npm run build

npm run app



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
PS C:\Users\Akanksha\Downloads\Dexter-s-Blockchain-main\Dexter-s-Blockchain-main> npm install

added 182 packages, and audited 183 packages in 36s

found 0 vulnerabilities
PS C:\Users\Akanksha\Downloads\Dexter-s-Blockchain-main\Dexter-s-Blockchain-main> npm run build

> svelte-app@1.0.0 build
> rollup -c

src/main.js → public/build/bundle.js...
created public/build/bundle.js in 1.6s
PS C:\Users\Akanksha\Downloads\Dexter-s-Blockchain-main\Dexter-s-Blockchain-main> npm run app

> svelte-app@1.0.0 app
> electron .
```

LANGUAGES & FRAMEWORKS USED

For Frontend:

1. Svelte
2. HTML
3. JavaScript

For Backend:

1. Electron
2. Node.js

Operating Environment

This application is highly flexible; it can run on all operating systems, including Windows, Linux, macOS, and can be executed on all computers capable of running server-based web applications powered by Node.js. It is also compatible with all popularly used web browsers

VIRTUAL TOUR OF THE PROJECT

Dexter's Blockchain

Group : G07

Assignment 1, Group: G07

Abhishek Khurana [2018A3P50621H]

Akanksha Vinod Hublikar [2018AAPS0317H]

Harshit Bansal [2018B5A70601H]

Dexter's private key

private

Rs 250

from Dexter to ▼

test

Add Transaction

► Sat, 25 Sep 2021 09:05:03 GMT | Rs100 | customer

Name1 → Dexter

▼ Sat, 25 Sep 2021 09:33:46 GMT | Rs250 | Dexter → test

Time : Sat, 25 Sep 2021 09:33:46 GMT

Amount : 250

Sender : Dexter

Reciever : test

nonce : 4371

Previous Hash :

000b01dd7c58b7a330960833f436e556afb3445ab2
20a319bb196b156eefffb

Current Hash :

00083fa3e35792a3b5f879a416580d406071dbc8e33
085925368021a54c5a5d0

RSA Signature :

1b523f784b1d5b4a9edd0bd5eb31b68b2236e2ceb2
a73e697de562e5d2bf9b442b87e2b0dbf77f2d9d613
bf47d1eedfdc47c10485be8c27e031c3b99c11c8d84
8b6a7b74e6d0eb4815065646837a950b042f1cef2fb
c88dc876fa59b9e54515909e5e968d312f2629a2d94
889ddd20fbbeb4e4a5d03d7c5027c6bd00c8def2b7c

Logic Behind the Working

Dexter(Business Owner) has a private key that only belongs to him. Only a user having this key is allowed to add transactions into the system. Every time a new block is created, the transaction data including, amount, previous block's hash, time of creation, and the private key together, are used to create a new hash for this block. To verify if the blockchain is valid, we only need to check if the last block's current hash matches with the current block's previous hash as the other blocks in the chain have already been verified before.

Encryption

The crypto module of Node.js has inbuilt functionalities that allow us to generate a hash for data. A SHA256(Secure Hash Algorithm) hash is generated using the following lines of code:

```
const generateHash = (obj)=>{  
  return crypto.createHash('sha256').update(JSON.stringify(obj)).digest('hex');  
}
```

This data and the private key that solely belongs to Dexter are used to create an RSA signature. We first convert the private key to a string and use this along with our data object to create a unique RSA signature with the help of the **createSign** method of the crypto module.

To verify if the data hasn't been tampered with, we check if the public key corresponds to the private key used while creating the signature for a particular block. This is done using the **createVerify** method of the crypto module.

Block

Each block consists of all the data corresponding to a transaction: the sender and receiver(Here Dexter can be either of them, as we are keeping an account of all the transactions both to and from the coffee shop) and the amount of the transaction. All this information, along with the private key, is used to generate the RSA signature of the block. This signature and the above data that includes the time of creation of the transaction and the previous block's hash would constitute all the information stored in our block. The entire data in the block is used to create a hash for it.

To verify if the block created is valid, we do two checks:

1. Verify if Dexter himself has created this block. We use the `verifySignature` function from 'Encryption' to check if the public key corresponds to Dexter's private key used to obtain the signature using the block's unhashable data.
1. Verify whether the current block's hash is valid. We use the `generateHash` function from 'Encryption' and compare it with the current hash stored in our block.

The most crucial issue that blockchain tackles is security. If the time required to create a block is significantly less, a hacker can tamper the block's data and recalculate the hash of all blocks after it, which could pose a severe threat to security. To avoid this from happening, blockchain has a concept of proof of work, which ensures that creating a block is computationally intensive. While mining a block, we make sure that the generated hash starts with a constant number of zeroes, defined as proof of work zeroes, 3 in our case, this number could be increased based on the requirements. This

function keeps regenerating hashes using a nonce, which is incremented by 1 for every time this loop is run and is also part of the hashable data for a block, so we generate new hashes every time the loop is run. Only when the generated hash satisfies the pow condition, we declare that the block has been mined.

Blockchain

To create a blockchain, we maintain two arrays; blocks and transactions. Here transactions contain the unverified blocks, and blocks include our verified blockchain. We now do the following three steps after the addition of each block:

1. To add a transaction, we push the sender, receiver, and amount information as an object into the transaction array.
2. To verify if all blocks in the blockchain till now, we do the following two checks for each block:
 - a. We use the verifyBlock function from 'block,' which verifies the block's author and the hash.
 - b. We also check if the previous hash for each block matches with the current hash for the last block
3. If the verifyBlocks function from the previous step returns false, we reinitialize the arrays blocks and transactions and update again.

Handling transactions: Every time a transaction is added to the blockchain, it gets stored in the transaction array. We run a loop iterating through the transaction array that obtains the prev hash of the newly to be added block from the block array.

```
let prevHash = '0000000000000000000000000000000000000000000000000000000000000000';  
if(blocks.length > 0)prevHash = blocks[blocks.length-1].unhashable.hash;
```

We now create a block using this prevHash and the other information. We then verify the author, mine this block, and add it to the blocks array, which is our blockchain. Since these blocks are now verified, they are removed from the transactions array. We also update the section in our application that displays these transactions at a regular interval using setTimeout.

CONCLUSION

The aim of the application is to make Dexter's coffee shop more efficient by making it easier to keep his transactions secure. It has a simple user interface, so even those with only rudimentary computer skills would be able to use it. The user interface is straightforward, easy to use, and self-explanatory. Our application satisfies all the requirements and also includes an interactive UI as an extra feature.

REFERENCES

Following are the online references along with the links that we have used for our project.

1. For Svelte-
<https://svelte.dev/docs>
2. For crypto module-
<https://nodejs.org/api/crypto.html>
3. MDN and W3 schools for documentation
<https://developer.mozilla.org/en-US/> <https://www.w3schools.com/>