NoteTogether: A Novel Distributed Note Taking Platform with Blockchain

James Kerr, Max Rioux, Tomas Surna, Fabio Marcellus, and Cuong Pham
School of Computing and Data Science
Wentworth Institute of Technology
Boston, MA 02115, USA
{kerrj, riouxm, surnat, marcellusf, phamc5}@wit.edu

2021 IEEE MIT Undergraduate Research Technology Conference ((URTC)) | 978-1-6654-0595-9/21/$31.00 ©2021 IEEE | DOI: 10.1109/URTC54388.2021.9701651

Abstract— **NoteTogether [1] seeks to bring interactivity to a very crucial aspect of online learning: consuming video media. We developed a hybrid distributed platform that provides a shared space for watching and annotating video media. Analytics are provided to all users to highlight how users are interacting with a video. NoteTogether utilizes Ethereum Blockchain to ensure data security and scalability by offloading data storage and processing requirements to the distributed Ethereum network.**

Keywords— Video, Annotation, Note Taking, Blockchain, Ethereum, Smart Contract.

I. INTRODUCTION

The global pandemic brought large-scale changes to education. Students around the world booted up their computers and attended class in online classrooms. While restrictions have lifted and students are returning to campuses, the popularity of online learning has grown significantly due in part to the tools that an online classroom can provide. Of note is the use of video media in an online learning space. In 2006, Taralynn Hartsell and Steve Yuen of the University of Southern Mississippi wrote, "[Video media] brings courses alive by allowing online learners to use their visual and auditory senses to learn complex concepts and difficult procedures" [2]. With the growth of streaming services like YouTube and Netflix, the breadth of educational video media available has skyrocketed.

Online classrooms provided a means of learning while the world was stuck at home, but online learning is not a lossless alternative to in-person learning. Principle among the issues raised by the online classroom is interpersonal interaction. The online classroom suffers from a disconnect between students, their teachers, and their peers. In a study performed by NUST Pakistan, 78% of students agree that face-to-face contact with the instructor is necessary for learning [3]. To improve the online classroom experience, the gap between student and teacher must be bridged.

The solution is NoteTogether, a distributed note taking platform that brings student-teacher interaction to the video-viewing experience. NoteTogether allows users to upload videos to Ethereum Blockchain and take notes together in a shared space. Viewership and note-taking analytics are captured and processed to display graphs detailing how users are interacting with a video. NoteTogether is a novel solution in video note taking space that utilizing Blockchain to store important data, ensuring data security and enabling scalability.

A. *Novelty*

MoocNote [4] is a similar application that provides users with the ability to take notes in a shared workspace and view analytics on viewership. However, there are key differences that allow NoteTogether to stand out:

1. MoocNote does not allow raw video files to be uploaded. This is an advantage of utilizing an InterPlanetary File System (IPFS) [5]. By customizing the size of nodes stored on the IPFS blockchain, NoteTogether can store a variety of video sizes while minimizing unused space.
2. NoteTogether requires users to pay a small fee of Ether when uploading to run blockchain interactions, but does not monetize. MoocNote monetizes via embedded ads and features locked behind a premium membership.
3. MoocNote and NoteTogether have differing goals. This can be seen in the analytics provided. NoteTogether provides analytics on all interactions with a video to provide insight on how the video was received and where points of interest may be located. MoocNote provides analytics on how individual users interact with a video to provide insight into who is engaging and how. MoocNote's goal is to support user accountability while NoteTogether's goal is to support user interactivity.

B. *Data Security*

NoteTogether's data is stored in three locations, Infuria IPFS [6], Ethereum Smart Contract [7], and cloud-based MongoDB [8]. These databases fall into two categories. First are the blockchain-driven databases. Both IPFS and the Smart Contract are run on blockchain, an immutable ledger. Blockchain maintains data integrity and validity by distributing copies of the blockchain across many trusted systems, and any changes must be verified via consensus of trusted parties. NoteTogether utilizes blockchain's protections by storing user-defined data. IPFS handles a wide range of data types and sizes efficiently, so video data is stored. The Smart Contract handles structured data of a uniform size efficiently, so note and user data is stored. The cost of utilizing blockchain comes in the form of transactions. All actions performed on the blockchain must be sent via transaction and incur fees. Transactions take time to mine which can bottleneck a system if many transactions are occurring at once. Transaction fees scale with the number of actions and processing required.

The final set of data NoteTogether must store is analytics data. Analytics data is not suited for blockchain storage for two reasons. First, this data is generated frequently which will incur large transaction fees and system throttling. Second, this data is not user-defined and therefore does not require the protections blockchain provides. For this reason, analytics data is stored in a cloud-based MongoDB database. Through the MongoDB Atlas API, MongoDB provides IP whitelisting to ensure analytics data is only added by NoteTogether.

C. *Scalability*

By utilizing blockchain rather than a traditional database, NoteTogether avoids concerns of storage scalability. Where a traditional database has a set size and expanding that size is costly, blockchain is effectively infinite. The cost involved in blockchain storage comes from transaction costs. The larger the data looking to be stored, the more expensive the transaction. For this reason, NoteTogether does not cover the cost for uploading videos. The user will cover the cost, paying more for a large video file. True scalability would see the user paying for posting notes, but this resulted in a bad user experience. Users looking to add many notes need to confirm many transactions with negligible transaction fees.

Where modifying data on blockchain requires a transaction that incurs fees and takes time to mine, reading from the blockchain is free and nearly instant. By handling user interactions in blockchain, any number of users can view the data without causing stress to the system.

The rest of this paper is organized as follows: Section 2 outlines the system design with our implementation details. Sections 3 and 4 presents our findings and potential improvements respectively. Finally, we conclude our work in Section 5.

II. SYSTEM DESIGN

Systems architecture is split into 6 components: The React user interface, the Ethereum smart contract, the Infuria IPFS directory, the MongoDB database, the Analytics Calculation Engine, and the Azure Scheduler. The React UI handles outward facing functionality such as the importing of new media and videos, notes, and analytics. Gathered data is distributed throughout the remaining components for storage and calculation. Video media storage is handled by the IPFS directory. Uploaded videos will be sent to IPFS in exchange for a hash key for future access. Video media metadata is stored on the live Ethereum smart contract. Data stored on the smart contract includes the video's IPFS hash key, UI access link, notes, user data. To calculate analytics for display, analytics data is stored on the MongoDB and processed in the calculation engine. The compute engine is a Kotlin server and will utilize the Azure scheduler to run calculations on a set timer. Once calculations are run, the results are sent to the UI for display.

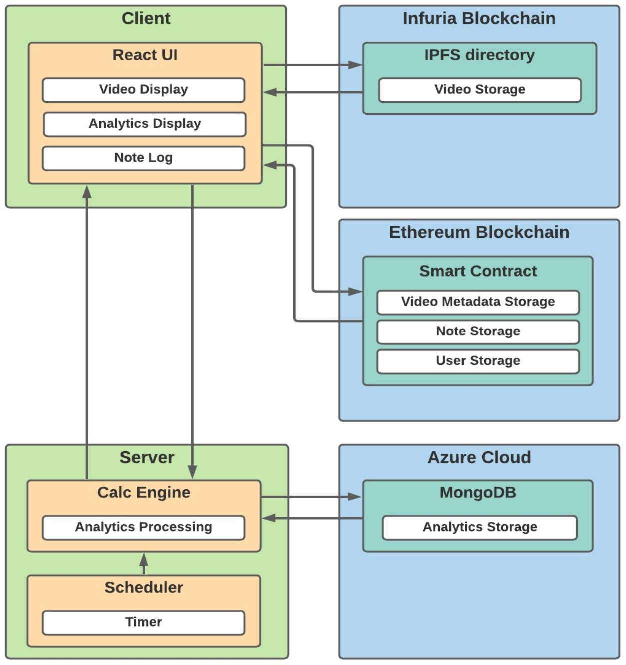
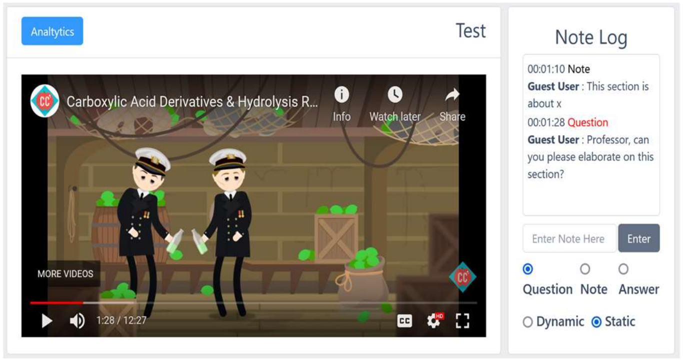


Figure 1. System Architecture

A. *Frontend*

The user interface (UI) is handled in ReactJS [9] framework. CoreUI [10] is a React UI component library that was used to build the pages and components of the web application to ensure quality UX and styling. Three pages handle the majority of features. These pages are the home page, the video page (Fig. 2), and the analytics page (Fig. 3). The home page utilizes MetaMask[11] , a browser extension that provides Ethereum Wallet integration and transaction generation. MetaMask allows the user to connect their Ethereum wallet to NoteTogether and they will be shown a form for uploading video media. Once submitted, the user is prompted by MetaMask to confirm a transaction at a small cost. After the transaction returns, the user is sent to the video page. Users can change their username by clicking on the profile icon and changing the username will also prompt a transaction. The video player was implemented using the react-player [12] library. React-player provides features for gathering analytics which are sent to the Analytics Calculation Engine for storage and processing. The video page allows users to view video media and add annotations. Annotations can be tagged as "Note", "Question", or "Answer" and are tied to the current video timestamp when posted. Posted notes are sent to the Ethereum smart contract using the Web3 API [13]. The note log displays all users' annotations statically, with all notes listed at once, or dynamically, with notes displayed by timestamp as the video plays. Two types of analytics are generated based on user actions: note analytics and view analytics. Note analytics are generated upon submission of a new note. The tag, timestamp, and date and time of analytic creation are saved. View analytics are generated when the user starts and stops the video player. The timestamp the user started watching, the timestamp the user stopped watching, and the date and time of analytic creation are saved. Saved analytics are sent to the Kotlin Calculation Engine for storage in MongoDB.



00:01:10 Note
Guest User : This section is about x
00:01:28 Question
Guest User : Professor, can you please elaborate on this section?

[✓ Question] Note Answer

Dynamic [✓ Static]

Figure 2. Video Page User Interface

The analytics page was implemented using the Chart.js [14] library, and uses a wrapper called react-chartjs-2 [15] for the graph's design. Displayed on the page are two graphs. First is a line chart representing annotation concentration by note tag. Second is a heat map that represents annotation and viewership concentration by timestamp. Together these graphs provide the user with a visualization of how the user base interacts with the corresponding video.

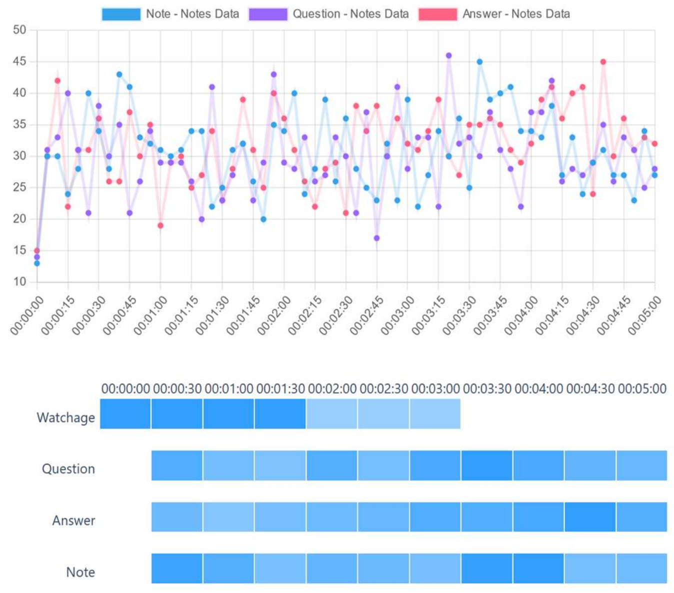


Figure 3. Analytics Page User Interface

B. *Smart Contract*

NoteTogether handles storage of video metadata, note data, and user data on Ethereum's blockchain. The NoteTogether Smart Contract is a Solidity [16] program that is pushed onto the Ethereum blockchain. Functions defined by the smart contract can be called by sending a transaction to the contract's address with function ID and parameters. The cost of the transaction is paid to Ethereum miners who perform the necessary computation.

C. *Infuria IPFS*

NoteTogether utilizes Infuria's IPFS for storage of video raw data and video links. IPFS allows for large data to be efficiently stored on blockchain while avoiding data duplication. IPFS generates nodes for data storage that can be customized to fit a given video size. This is important given that videos can either be uploaded as a full video file or by link.

D. *MongoDB*

Storage of analytics data is too costly on Ethereum Blockchain due to its large quantity and small size, so NoteTogether utilizes an Azure [17] Cloud-based MongoDB database for analytics storage. MongoDB provides flexible storage to handle a wide variety of data quantities.

E. *Analytics Calculation Engine and Scheduler*

The Analytics Calculation Engine is a Kotlin server with two functions. First, the engine acts as middleware between the frontend and MongoDB database. Second, the engine processes analytics data stored in the MongoDB database for display on the frontend's analytics page. An Azure Scheduler [18] is used to rerun the calculation engine to update the page after a set time has passed.

III. RESULTS

NoteTogether utilizes various external components. To determine the scalability of NoteTogether, components were tested to determine potential bottlenecks. Testing focused on two types of database interactions that may affect user experience as NoteTogether grows.

First is blockchain transaction speed. The cost of securities provided by blockchain is the transaction mining process. Transactions must be chosen by miners, who choose which transactions to mine based on transaction fee. When a transaction is chosen, the miner will perform all actions required and add the transaction to a new block. When the block is added to blockchain, the miner is rewarded by receiving the money from the transaction fee. For this reason, the transaction fee may greatly affect the speed at which a transaction is chosen for mining. This speed will affect how fast a video is uploaded or how soon a note will display on another user's log. Transactions calling the same smart contract's functions must be performed in the order submitted. This could allow a single slow transaction to halt video and note uploads for all users.

Second is the Kotlin Calculation Engine speed. While the speed at which analytics are added to the MongoDB has little effect on the user experience, delays in analytics display will show the user a loading screen. It is important to determine the effect that a big dataset will have on calculation engine processing times.

A. *Blockchain Transaction Speed*

TABLE I. Transaction Mining Time By Gas Price

Avg Gas Price Multiplier	**Mining Time**
x0.5	1.5 hr
x0.95	16 min
x1	20 sec
x1.5	17 sec
x2	21 sec
x5	8 sec
x10	19 sec

To ensure miners are paid appropriately for work done, transaction fees are determined by the amount of work performed. The amount of work performed is quantified as "gas". When choosing a transaction fee, the amount of gas required does not change, but the price of each unit of gas can change. Gas price determines the monetary gain of a miner for mining a transaction. Average gas prices throughout the blockchain network will fluctuate based on network demand. Web3 [13] provides functionality to retrieve the average gas price from the last ten mined transaction on the network. To determine the effects of various gas prices, note submission transactions were submitted with various multipliers applied to the current average gas price. Mining times were calculated by timing transaction response through the UI and by tracking the transaction in Etherscan [19]. For blockchain testing, NoteTogether was deployed on the Ropsten Test Network. Notably, gas prices below the average, even by a small margin, drastically increase mining times. Gas prices above the average, however, provide little change to mining, and fluctuate in a way that is likely caused more by luck in transaction submission timing than gas price.

To avoid wasting Ether, the optimal gas price is the average gas price as defined by the last ten transactions. Table I suggests that attempting to shorten runtime by increasing gas prices proves ineffective. This is likely due to the economics of Ethereum mining. If a miner waits for a transaction with a higher payout to mine, they waste time where they could be mining a transaction. The transaction with a high payout might be snatched up by a miner who is less picky. It seems that few miners are less picky than the average transaction price, so there is likely a drop off in profits to mine below the average transaction payout.

B. *Analytics Processing*

The Kotlin Server handles the calculation of analytics data for display. To test the scalability of the server's calculation engine, the MongoDB was stored with analytics in a range of quantities from 10 to 30,000 analytics stored. An analytic is a recorded action, either a note added or a duration of the video watched, that is marked with a relevant timestamp. The calculation engine was then run and timed. The engine must retrieve all stored analytics and processing through the calculation engine. The processing times below include both retrieval time and calculation time. Each value was gathered 3 times and averaged. Processing times are internal to the calculation engine to avoid network irregularities.

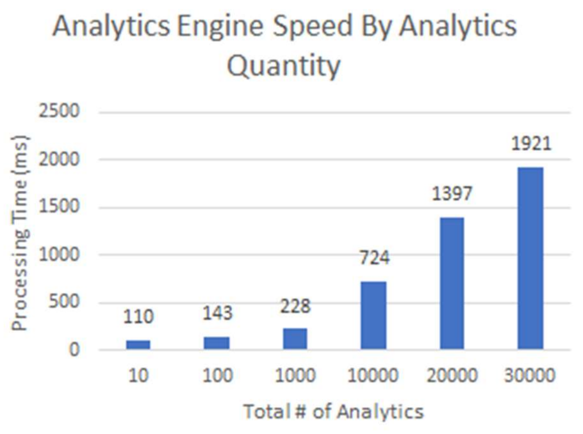


Figure 4. Calculation Engine Speed Graph

Figure 4 represents the processing time of the analytics calculation engine based on the quantity of stored analytics. While the processing time grows substantially from 10 analytics to 30,000 analytics, the time stays within the range of a few seconds. Figure 5 proves processing time and input quantity are not proportional. Figure 5 displays the processing time in milliseconds per analytic. As the quantity of stored analytics increases, each new analytic has less of an effect on total processing time. This can be explained by examining MongoDB's Find network call. When the Kotlin Server uses the Find call, two general actions are performed. The server connects to the database, and the database processes the request. No matter how many items are retrieved, the server still must connect with the database. The difference is the ratio of work expended by the connection action to the retrieval action. As the number of items increases, the weight of the connection action decreases, thus a skew is generated. Thus, increasing the analytics quantity is expected to have a diminishing effect on processing time.

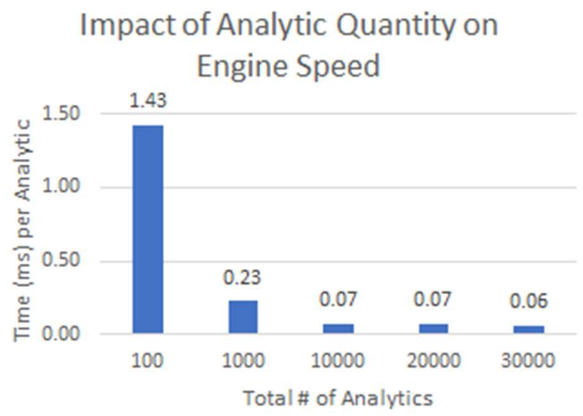


Figure 5. Graph of the Impact of Analytics Quantity on Calculation Engine Speed

To determine whether the calculation engine would create a bottleneck, calculation times must be compared against their effect on the user experience. A two second delay to display analytics is acceptable, but a ten second delay may raise complaints. By projecting the results of Figure 4, it is estimated that over 100,000 analytics generated on a single video must be added to exceed a five second delay. This estimation far exceeds the expected usage of a typical online classroom. It is unlikely for the Kotlin Calculation Engine to become a bottleneck as NoteTogether scales.

IV. FUTURE WORK

A. Grouped Note Transactions

Concerns have arisen around note transactions. Each note a user submits is handled in its own transaction. For this reason, costs of note transactions were covered by the NoteTogether contract to provide a better user experience. If we were to hold note transactions until the user submits all they wish to add, transaction quantities would be decreased significantly, and note costs could be handled by the user without negatively affecting the user experience.

B. Custom IPFS Network

The Infuria IPFS network provides a hard limit to maximum node size which in turn limits the maximum size of videos that can be added. This limit can be bypassed by utilizing a private network. The downside of a private network is in joining and using the private network, which would increase costs.

C. Ethereum Alternatives

Ethereum has seen a large boom in usage and price in the past few years. The increase in network demand has led to an increase in gas prices. To lower costs, NoteTogether could switch to a smart contract system built on a cryptocurrency that is in less demand. Ethereum 2 is in development which will forgo the slow proof-of-work mining methods in exchange for a much faster proof-of-stake method. Coming Smart Contract networks plan to implement sharding, a system of interweaving multiple blockchains to improve scalability. Alternatively NoteTogether could pivot to an existing cryptocurrency with a lower network demand. Options include EOS [20], Cardano [21], and Polkadot [22].

V. CONCLUSION

NoteTogether provides users with a shared platform for annotating video media that differs from the competition by utilizing the security and scalability of blockchain. NoteTogether offers its services to the user at a cost based on usage. User data is stored securely without the need for database expansion. By utilizing blockchain's distributed network of miners, NoteTogether avoids many system bottlenecks. However, blockchain introduces a crucial bottleneck, the transaction mining speed. NoteTogether is only as fast as its transactions, and transaction mining speed is determined by network demand. The progression of NoteTogether would see exploration into transaction efficiency. NoteTogether will be looking at coming breakthroughs in transaction speeds such as proof-of-stake and sharding to relieve transaction bottlenecking and truly realize the scalability provided by blockchain.

VI. REFERENCES

[1] NoteTogether: https://github.com/tomassurna/NoteTogether
[2] Taralynn Hartsell, Steve Chi-Yin Yuen, The University of Southern Mississippi, United States
AACE Review (formerly AACE Journal) Volume 14, Number 1, January 2006 ISSN 1065-6901 Publisher: Association for the Advancement of Computing in Education (AACE), Waynesville, NC USA
[3] Adnan, M., & Anwar, K. (2020). Online Learning amid the COVID-19 Pandemic: Students' Perspectives. *Online Submission*, *2*(1), 45-51. https://eric.ed.gov/?id=ED606496
[4] MoocNote: moocnote.com/
[5] IPFS: ipfs.io/
[6] Infuria: infura.io/
[7] Ethereum: ethereum.org
[8] MongoDB: mongodb.com/
[9] ReactJS: reactjs.org/
[10] CoreUI: coreui.io/
[11] MetaMask: https://metamask.io/
[12] React-Player: github.com/CookPete/react-player
[13] Web3: https://web3js.readthedocs.io/en/v1.4.0/
[14] Chart.js: github.com/chartjs/Chart.js
[15] React-Chartjs-2: github.com/reactchartjs/react-chartjs-2
[16] Solidity: soliditylang.org/
[17] Azure: azure.microsoft.com
[18] Azure Scheduler: https://docs.microsoft.com/en-us/azure/scheduler/scheduler-intro
[19] Etherscan: https://etherscan.io/
[20] EOS: eos.io/
[21] Cardano: cardano.org/
[22] Polkadot: polkadot.network/