Yet Another Blockchain-based Privacy-friendly Social Network

Lars Andreassen Jaatun, Anders Ringen and Martin Gilje Jaatun
IDE
University of Stavanger, Norway
lars@jaatun.no

***Abstract*—Current social media platforms suffer from the fact that ownership of personal data is transferred to the owner of the network the moment you post it. This paper demonstrates that it is possible to create and maintain a social network using Hyperledger Fabric, where personal data is protected from users that should not have it, and all operations on data is recorded in the ledger, so you can audit how your data has been used by others, even the owner of the social network.**

***Index Terms*—Social network, security, privacy, blockchain, DLT**

I. Introduction

Social media today allows you to upload, share and comment on content of all kinds, whether it be a famous actor's bad take on proper etiquette or the latest arrest by your local police. When you post something on a social media platform, you essentially donate a whole load of personal data to the owner of the social medium. After you post something, there is no telling what happens to the data. Even though a user of a social network should own their own data, ownership is transferred to the owner of the network the moment you post it.

Our solution [1] uses blockchain technology to enable the tracking of user information. There are aspects of blockchain that can provide solutions to problems with regular centralized social apps. Blockchain technology can verify the origin of data, and restrict unauthorized access. This is due to the fact that all operations on assets stored in the blockchain are recorded in the ledger as transactions, and can therefore be audited. We have created chaincode that can interact with the ledger to read both historical information and the current state of the ledger. Ledger data is fetched to a mobile application written in react-native through an API that interfaces with the network. This enables users to track how their data travels through the network, and who has access to your posts.

The remainder of this paper is structured as follows: Section II describes relevant background, and Section III explores existing approaches to our challenge. Section IV describes the chosen project design. Section V is an evaluation of the final product, and Section VI discusses the choices and strategies employed in the project. Section VII proposes further work, and Section VIII concludes the paper.

II. Background

A. Smart Contracts

Within blockchain technology, one of the most attractive points for business applications are smart contracts. They enable businesses to write code instead of physical contracts, and these coded contracts are automatically fulfilled when conditions agreed upon by both parties are fulfilled. This has the potential to decrease costs for both parties, as well as time spent on bureaucracy. This however is not the only use case for smart contracts [2].

As well as being used for legal contracts between corporations, smart contracts can be applied in numerous fashions within blockchain networks to run distributed applications on multiple clients.

Smart contracts are not limited to transferring assets (e.g. cryptocurrency) from one party to the next, but can also be used to change states of programmatic objects defined within an application. A good example of this can be found in the hyperledger fabric documentation [3]. They describe a car ownership smart contract. In the application, there are multiple cars owned by different people, person A and person B. Person A initiates a transaction to buy a car, and person B initiates a transaction to sell this car to person A. The smart contract takes the inputs and completes the trade by changing the state of the car object to reflect that it was sold to person A, all without making any physical transfers between users of the application.

B. Hyperledger Fabric

Hyperledger Fabric (HLF) is an open source project created to enable blockchain technology for use in a business context with strict requirements on privacy, confidentiality and auditability. These concepts are traditionally difficult to combine within the same project. Nevertheless, the project turned out to become quite successful [4]. Unlike most cryptocurrency blockchain networks, HLF is a permissioned network, requiring participants to be authenticated to join. HLF networks are built up of channels. Channels are separate sections of the network with their own ledger and their own members independent of the other channels in the network. Different members have different privileges depending on the role they have been assigned in the network, such as administrators with permissions to change configuration options and commit transactions, or members that have read-only access. Nodes that act as access portals to the channels for the members are known as peers, and they too have roles defined by the configuration that differ from the roles given to users.

1) Configuration: HLF is configurable to a large degree. One HLF network will most likely be different from any other network you can find. The configuration options range from endorsement policies within a channel, to orderer configuration, whether there are many orderers or a single orderer per channel, and how many channels you want for your network. The configuration isn't static however; if you have the right permissions (i.e., you are a channel administrator) you can send a proposal for a configuration change, and the change will be accepted depending on the existing policies.

2) Endorsement Policy: The endorsement policy describes which members from which organizations in the channel must "endorse" or approve smart contract transactions [5]. If the transaction is not endorsed as required, the transaction is rejected.

3) Channel Configuration: The channel configuration is stored in a collection of transactions that is normally abbreviated as configtx [6]. The channel configuration contains settings for the organizations present in the channel, for the peers that will participate from the different organizations, as well as configuration for the ordering service in the channel.

4) The Ordering Service: Permissionless blockchain networks typically have consensus mechanisms that not only require the participation of a large proportion of network members, but also take time, are very power-hungry, and typically result in some nodes doing a lot of work for no payoff. HLF instead uses orderer nodes for ordering and assembling blocks that can be added to the blockchain. A group of orderer nodes form an ordering service, that in addition to assembling blocks to the blockchain manage basic access control for channels, and enforce policies for validation of transactions created by the administrators of the organizations participating in the network [7]. When the ordering service receives enough transaction proposals that are endorsed according to the endorsement policy to fulfill the minimum requirements to constitute a block, it packages it into a block which is then distributed to all peers in the channel. Using an ordering service instead of using all peers for endorsement and ordering of transactions results in higher throughput of transactions, as well as completely avoiding forks in the ledger. As with all other nodes, orderers are owned by the organizations participating in the channel.

HLF supports multiple protocols for consensus among multiple ordering nodes, but recommends the use of the Raft protocol, due to it being simpler to set up and manage for network architects [7]. The implementation is crash fault tolerant, which means that the service can reach consensus even if some orderer nodes fail, but it is not Byzantine fault tolerant [8], which means the service cannot reach (reliable) consensus with malicious nodes in the ordering service. Therefore one should avoid sharing a channel with untrusted organizations.

5) Identity: Identity in HLF is most commonly managed using digital identities in the form of digital certificates issued by a trusted certificate authority (CA). Fabric provides binaries for a certificate authority called fabric-CA. Within channels, there are membership service providers that decide what certificates have access to the ledger. The HLF documentation likens the relationship between certificate authorities and membership service providers to credit cards and credit card terminals, respectively. The example is that some card terminals may accept only Visa, while others only American Express, and others again may accept both American Express as well as Visa [9].

6) Smart Contracts in HLF: HLF distinguishes between smart contracts and chaincode in the documentation [3]. Chaincode is used interchangeably with the word smart contract, but they have slightly different meanings. A smart contract is a collection of functions that carry out terms agreed upon by all parties participating in the transaction. A chaincode on the other hand is a package of multiple smart contracts, and is what you deploy to a channel.

III. Existing Approaches

There are currently multiple social networks that tout a blockchain-backed technology stack. Steemit [10], Somee [11] and Sapien [12] are examples of social networks that are all based on blockchain technology. While Steemit and Sapien seem to be reputable, or at the very least have good intentions, there is evidence pointing towards Somee being a scam [13]. This is also important to showcase because of the prevalence of cryptocurrency related scams based on encouraging large investments into the product while promising huge payouts, only to liquidate their own (often substantial) stake, causing the value to drop significantly. Steem, Sapien and Somee all have the same focus which is the monetization of content through their own cryptocurrency.

Both Sapien and Steemit boast a virtually censorship-free social network as an alternative to censorship-heavy centralized platforms. However, in their respective white papers outlining the technology stack, there is barely a mention of privacy, or a user's right to their own data [14], [15]. While Sapien's marketing lead of 2020, Jonathan Goodwin, wrote a piece on how users are entitled to their own data [16], the piece is accompanied by a disclaimer that clarifies that the piece does not reflect the official position of the Sapien social network, even though it is written to sound like it is. So while the platforms may better accommodate for free speech, users still do not own their data.

Ushare is a proposition for a decentralized social network [17]. This design also utilizes blockchain technology, but in ways that differ from the approaches described above. The most significant usage of blockchain technology here is not the monetization of social interactions, but not unlike our project the tracking and restriction of data sharing throughout the network. To achieve this, they create a token associated with each post uploaded to the network. On every sharing operation, this token decreases, until it reaches zero whereupon it cannot be shared further.

Ushare argues that the inherent properties of public blockchains like anonymity, resistance to censorship and inherent decentralization make then an excellent candidate for hosting a social network. The paper describes a system for distributed storage of larger files for predictable growth of the blockchain. This load-sharing technology would mainly be for video posts. To control the sharing process itself, the paper describes a "turing-complete relationship system", which is equivalent to a smart contract as described above. Ushare introduces interesting ideas that are applicable to our project. While storing the data outside the ledger might be good for performance, it could also be beneficial from a security standpoint in that it could make it possible to store data with the user instead of in a server that is publicly available.

IV. Proposed Solution: Just Us

Just Us will use Hyperledger Fabric to enable users to both claim a higher level of ownership over their own data, as well as determining the path their data travels throughout the network. The choice fell on HLF due to good availability of documentation, both official and third-party, as well as HLF being a permissioned network. While Hyperledger Sawtooth was also an option, we decided to go with Fabric due to not needing the flexibility of having both a permissioned and public ledger.

The project utilizes the test network provided by the HLF developer team. The test network contains two peer organizations and one ordering organization. It is deployed to an isolated docker compose network and is designed to test new applications and smart contracts.

For consistency, both the chaincode and any other components for the backend portion of the application should be written in Go. The application should expose an API that a mobile application can interface with, as well as an HLF network that communicates with the API.

The application should be able to track posts made by the users, and manage access control to different users' posts. How other users access a user's post should be recorded in the ledger as transactions, and access to data should be managed by the user that owns the data. Users should be able to retract another user's permissions to view their content, and otherwise accurately regulate access to their data.

Instead of storing all data on the blockchain, non-transaction data should be stored off-chain on an external database only accessible to the backend server. This creates more predictable growth of the size of the blockchain since all assets persisted on the ledger are of the same size, as well as making it easier to implement different types of content uploads such as photos or videos to the application.

Our solution uses a REST API together with an HLF network. The users again authenticate to the REST API using JSON web tokens, and access to the HLF network is done through an HLF gateway enabled peer using a single certificate authenticating the server. A single channel holds all users' posts, and access to a user profile is controlled through the API.

Assets on the ledger are profiles, each representing a single user. A minimal profile asset includes the data fields "username", "followers", "followed users", "pending followers" and "posts" (a list of the posts the user has either created or shared). The list of posts are data objects consisting of the fields "owner", "post id" and "sharing history". Sharing history is a list of users that have shared a post, and is used to verify whether another user has access to read the post.

A. Frontend Design

The requirements for the frontend were to be able to navigate from one screen to another, to allow for easy styling and shaping of components in the interfaceand to make API calls to the backend server. More specific functionality included having a home screen with posts of followed users, being able to follow and accept follow requests, being able to forward posts in a private chat between users, and create posts and users. Other proposed functionality was being able to see your own posts, track where they ended up and control it by restricting certain users access.

We used the React Native [18] framework because it has a good reputation, being developed by Meta [19]. Code written in React Native works the largely same for both Android and iOS, making deployment to the two major platforms a lot easier.

1) User Interface Design: Diagram 1 shows the proposed user interface design for the mobile application.

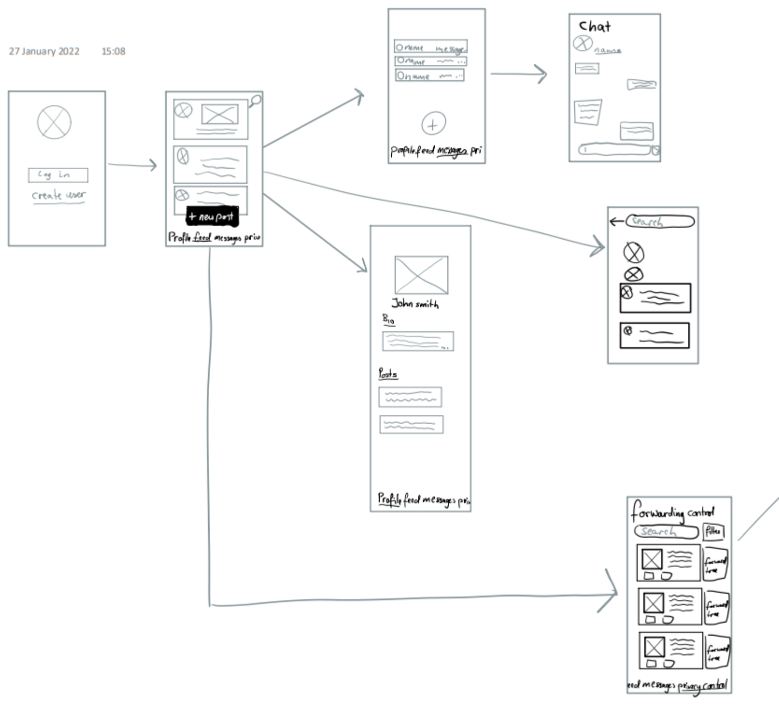


Fig. 1. UI flow diagram

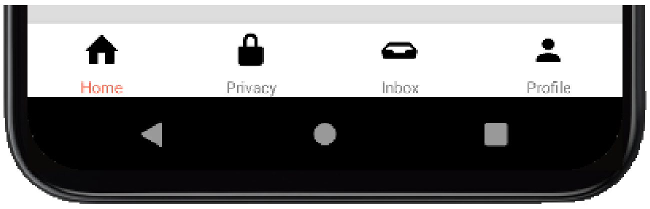


Fig. 2. The navigation bar on the bottom of the screen

B. *The First Screens and Tab Navigation*

The navigator is a bar on the bottom of the screen with four different touchable icons which navigates the user to its respective screen (Fig. 2). The stack navigator was used on the four tab screens to further navigate to other screens, stacking upon the original screen which gives you an arrow in the top left corner to return to the previous screen. The mobile application has support for instant messaging, however it wasn't implemented in the backend portion of the project.

C. *Nests and Stack Navigation*

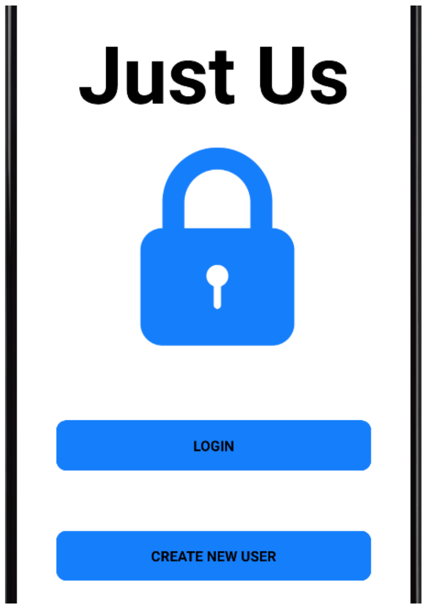
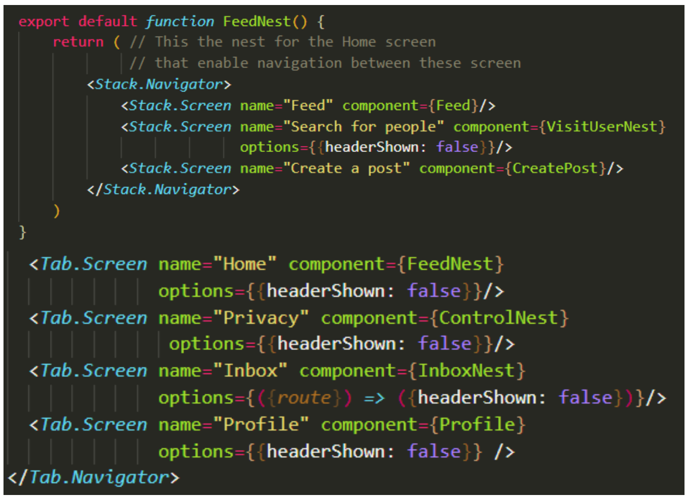


Fig. 3. The first screen of the application

We then made the index screen, which is shown in Fig. 3. This is the first screen the user sees when opening up the application. Nests were used to implement this. A nest is a navigation system within a navigation system. Most of the four tab screens got their own nest which had a stack navigation within the tab navigation, shown in the code of Fig. 4.

To make the navigation from the login screen to the main content of the application, a nest enfolding the four nests had to be made. This became a stack navigator within a tab navigator, within a stack navigator where you hide the return arrow when you navigate from the login screen to the main application. The login functionality was made by



```
export default function FeedNest() {
    return ( // This the nest for the Home screen
            // that enable navigation between these screen
        <Stack.Navigator>
            <Stack.Screen name="Feed" component={Feed}/>
            <Stack.Screen name="Search for people" component={VisitUserNest}
                          options={{headerShown: false}}/>
            <Stack.Screen name="Create a post" component={CreatePost}/>
        </Stack.Navigator>
    )
}
 <Tab.Screen name="Home" component={FeedNest}
             options={{headerShown: false}}/>
 <Tab.Screen name="Privacy" component={ControlNest}
              options={{headerShown: false}}/>
 <Tab.Screen name="Inbox" component={InboxNest}
             options={({route}) => ({headerShown: false})}/>
 <Tab.Screen name="Profile" component={Profile}
             options={{headerShown: false}} />
</Tab.Navigator>
```

Fig. 4. The upper image is the stack navigator for the home screen with the post feed, and the lower code shows that stack as the first component of the tab navigator, together making a nest.

globally storing the logged in user with AsyncStorage [20][1], and setting it to an empty string when logged out.

The screen displaying posts from other users is called the home screen, and can be seen in Fig. 5.

D. *Connecting with the API*

The frontend communicated with the backend portion of the application through an API. For all functions within the application, from following another user to finding a user on the network, we created API endpoints that took GET and POST requests respectively, depending on the action needed.

Making it possible for a user to accept the follow request was done by adding a button on the profile screen that displayed the list of all users that have sent a follow request. This list of users is obtained by fetching data from the "'/Profile?username='+user" endpoint that returns all data associated with the user, including its posts, followers and follow requests. With the help of the React hook(feature) Effect [21], the data fetch happens as you enter the screen and the list of pending followers is saved in a state variable. The display of the pending followers has a pressable green check symbol that accepts the follow request. This is done by sending a POST-request to "...:8080/User/AcceptFollow" together with the user and the future follower as data in the body.

Further development included adding a button on all posts displayed on the home screen. This was a share button that would have the functionality to share the post further to the followers of the user sharing the post. In simpler terms, when a user shares a post, it posts it to its own followers, with the original creator and content. This is by sending the user, owner and post ID to "...:8080/Post/Share'". This potentially exposes the post to

[1]AsyncStorage is now deprecated, so for future development it would have to be replaced with a maintained solution to avoid the application breaking upon a new update.

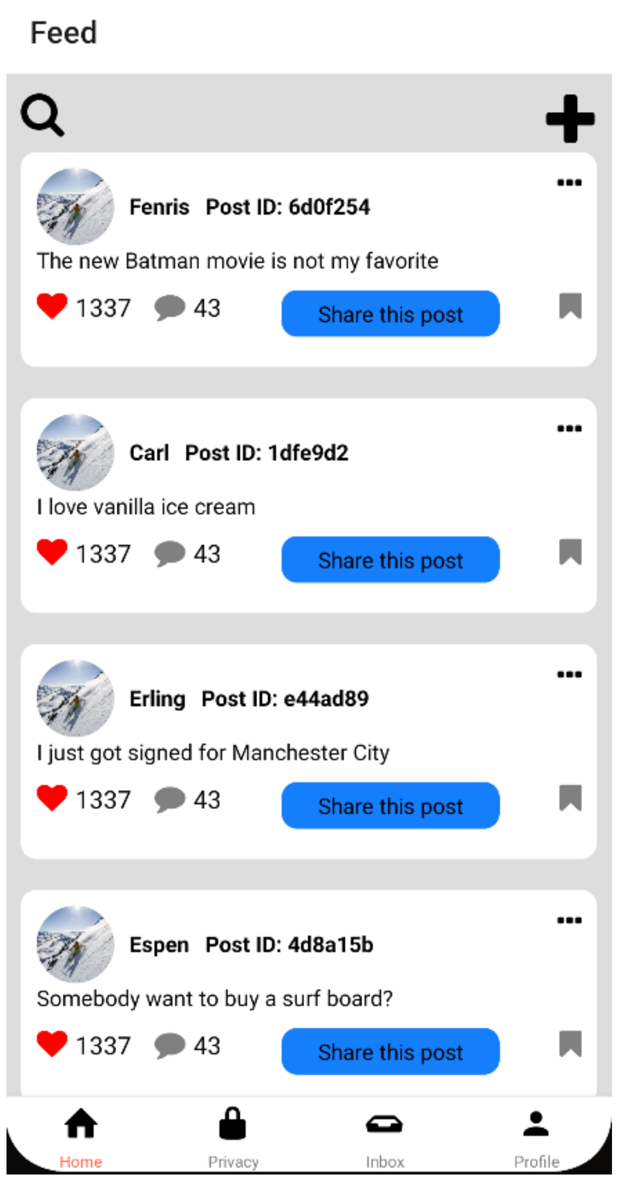


Fig. 5. The home screen with the post feed.

users not following the original creator, and giving them the opportunity to share the post even further.

E. Privacy

When there is a possibility to share posts, the original creator should know who shares the post and where it ends up. This is where the privacy screen comes into play. This screen also uses Flatlist [22] to display the post ID of a post and the users that have shared the post. This was done by first fetching all the posts of the logged in user, and sending the list of posts into another function called "makeShareList()". This function loops through the post ID of every post in the list and sends a "GET"-request that return the transaction history of the post, which includes every user that has shared that post. This data is pushed onto another list, as an object containing the sharers of the post and the post ID. This list is used when rendering the item that is displayed in the Flatlist [22] on the privacy screen. It includes conditional rendering, rendering different components depending on whether or not there exists any sharers of the post. The last step is distinguishing the sharers of the post that follow the original creator and not. This is done by fetching the follower list of the user, and checking if the sharer is included in the follower list. The sharers are then conditionally rendered with different colors. The followers of the original creator are displaying with normal black font color, and the sharers that do not follow the original creator will be displayed with red font color. This is done by giving the latter's "<Text>" tag its own style component.

F. Backend

The chosen solution consists of three main components: The chaincode, the chaincode tests, and the API. All components are written in Go for consistency.

1) Chaincode: The chaincode uses the smart contract API created by hyperledger fabric to facilitate creation of smart contracts [23].

All chaincode functions are methods on the struct SmartContract defined earlier in the script, as well as taking in the transaction context. Taking in the context like this makes the function more testable. Whenever functions in the chaincode need to access the ledger, they use the transaction context to pass the function getStub() which gives access to functions that operate on the ledger as well as the world state, in this case the function creates a query for the complete history of the profile with the corresponding userId. All queries are simple key-value queries.

2) Chaincode Tests: Tests for chaincode were a necessity for speedy development. The virtual machine used more than 2 minutes to rebuild the network and install new chaincode on the peers, which made it near impossible to weed out small bugs without having the tests.

The tests use mocks generated by counterfeiter to be able to pass a fake transaction context to the chaincode functions. As per usual, the tests aim to test for edge cases.

3) API: The final component of the backend is the API. The API creates the interface between the mobile application and the fabric network. The connection is made with a gateway enabled peer. To authenticate the API to the network, a certificate is created from one of the pre-made identities in the HLF test network. This identity is stored in a wallet folder in the same folder level as the server. The server uses the gorilla/mux [24] library to enable the passing of parameters in the url.

When contact has been made between the gateway enabled peer and the API, the gateway can be used to fetch an instance of the chaincode running on the network. from that instance, the API can call chaincode functions.

The call to the chaincode function FollowProfile is made through the chaincode instance. parameters must be passed as strings. The function followUser is called when the endpoint /User/Follow is hit.

G. Running Environment

The chaincode and API development was done on a remote virtual machine running linux. This was because HLF plays nicer with linux systems than other environments. In addition to this, using the VM provided us with a static IP address reachable from within the UiS network, thus we could easily host the website for the mobile application.

H. Frameworks/Tools Used

For development of chaincode and API we used the Visual Studio Code [25] programming environment together with its SSH extension. The API was written using the gorilla/mux [26] package to be able to fetch parameters from the URL. This is mostly a feature for quick prototyping, as passing parameters in the URL is generally considered unsafe.

The code for the finished product can be found on github in the repository at this link: https://github.com/UiS-DSComputing/just-us-blc-social

V. Evaluation

Execution Time Chaincode execution in HLF is very fast relative to public blockchains like Bitcoin, but still proved to be sluggish in our application. The likely cause for this is the rate at which the orderer was configured to assemble and verify blocks. If the amount of transactions do not reach the threshold for creating a new block there is a timeout that initiates the creation of a new block, resulting in slower execution.

Data Sharing Our application demonstrates the concept of tracking data based on querying the transaction history. To obtain the sharing path of the data object, the application gets the history of the post in question. Though the application does not use the complete history in its current iteration, there is the potential to automatically audit this history of events to see if there is something that does not match what the history of events should be. Using the transaction history is more valuable than simply checking a state variable, as a state variable does not provide a history of all their past states and thereby if access to the account is compromised it is more difficult to discover. The current implementation shows the user which of their followers who shared their post, and which users that do not follow the user who also shared the post further.

Query Efficiency HLF supports two different types of state databases for storing assets (in this case the profile objects): LevelDB and CouchDB [27]. Whereas LevelDB is faster, couchDB supports more advanced queries if the assets are stored in JSON format. The application only uses key-value queries for the application, which potentially could be improved using more specific JSON queries. While for the current configuration LevelDB seems to be more appropriate, as the chaincode only addresses the ledger using key-value queries, due to the complexity of the data object, more complex queries could speed up certain transactions that take a lot of time due to having to filter out the fields of interest instead of receiving them as raw data.

Network Design For network design the project largely relied on the test network provided by the HLF development team. This is not a problem for testing of smart contracts and developing a basic proof-of-concept. However, for experimenting with different policies and different network configurations and topologies, it is necessary to design a bespoke network. In the documentation for the test network, the HLF developers state that the test network is prone to breakage upon changes. This is most likely due to the accompanying script for deploying the network which automatically sets up all facets of the network, from creating a certificate authority, to setting up organizations, to creating channels for those organizations by hard-coding.

Data Uploads For the moment, data uploads are limited to text uploads. This does not need to be the case, as the data is not stored on the blockchain, only in the off-chain database. Therefore to enable the uploading of photos and/or videos, only small changes in the database as well as in the API to handle sending and receiving videos would need to be made.

Decentralization In its current state the application does not support decentralization. It would be possible to deploy the social network chaincode to multiple channels, however users in different channels are not discoverable to eachother. This is an inherent flaw with the design of the application.

Security The final design is not particularly secure, especially when it comes to non-repudiation. For instance, the use of JSON web tokens (JWT) could improve the security of requests to the network, however this was not implemented in the final product due to basic functionality being prioritized.

Scalability The solution approach likely does not scale well. While improving the design of the data objects could speed up the existing queries, in the end the approach with multiple users sharing one channel is not sustainable. Eventually with a larger user base, the application would need the kind of processing power that is available to other social media. One way to improve the scalability of the application would be to adapt the architecture towards a more decentralized solution. Decentralization could result in a network with better fault tolerance and availability, as well as keeping performance at an acceptable level.

Encryption Data is uploaded as-is without encryption to the database. This is not secure; if someone gains access to the credentials for database connection, they may compromise the privacy of the users. Enabling the encryption of posts is decidedly an important step for improving the security of the application. Posts could be encrypted using the keys distributed by the fabric CA.

VI. Discussion

At the moment, blockchain technology is still surrounded by a lot of hype. New non-fungible token frameworks and cryptocurrencies are being promoted by influencers of all types on many different platforms without ever presenting any novel ideas, while promising a huge payoff for those brave enough to invest. Because of the popularity of blockchain technologies like Bitcoin, the rate at which "blockchain" is suggested as a solution when either not applicable or when another solution may work better is most likely too high [28]. However, in the case of this project there is clearly potential. Many of the problems associated with social media are related to incorrect or unlawful treatment of personal data. With an HLF based system, it could all but eliminate the human component in the treatment of data, in the way that the only administrator of the data uploaded to the network is the creator, save for administrative rights to, e.g., remove channels and/or users that violate the terms of service for the application, as well as opening up for a decentralized approach where users do not depend on a single centralized service provider. It also opens the door for actually owning your own data, giving you the opportunity to host your own data and thereby have complete control over your data.

Traditional Social media earns revenue from selling information on its users, and from letting other companies display advertisement to a targeted audience. An application with stricter control over personal data will not have this income, which makes the maintaining of servers and other expenses harder pay for. Moreover, this could necessitate direct payment to participate in the network. As is apparent in the amount of users of social media today, "people in general" do not seem to care enough about their personal data for it to be possible to implement payment for participation.

The function that shows tracking information for a post is very slow, as well as not persisted, so whenever you want the sharing history for a post it has to be generated from the transaction history. A way to mitigate this would be to persist a graph data structure within the post object that maintains lists of nodes and edges, and is updated upon sharing operations. For security, this data object could be regularly audited using the transaction history for both the profile and the post in question. This would catch any malicious attempts at accessing content.

To avoid selling blockchain as the be-all and end-all when it comes to the treatment of personal data, there are some assumptions that must be addressed and avoided. Yes, sharing may be better controlled by the owner and yes, malicious access may be less of a problem, but using blockchain doesn't automatically mean that you own your data. **Just Us** does not collect and store user data for analysis in its current iteration and neither will it if development continues, but the other options for social media backed by blockchain technology do not necessarily adhere to the same principles. In practice, the usage of blockchain should be viewed with the same scepticism that something labeled as "water-resistant" gets versus something labeled "water-proof"; while blockchain may help with controlling user data, it is certainly not enough on its own. It must be combined with other technologies or principles so that everyone who uses the app can be certain that the application only consumes the data that it is intended to consume. One such design principle is open source. In the early stages of Elon Musk's proposed acquisition of Twitter, Musk claims that he will open source the algorithms used in Twitter for increased trust [29], thereby enabling individuals that hold the applicable competence to inspect how user data is treated by the network. Note that Musk never mentions anything about not collecting user data.

VII. Further Work

HLF has support for decentralization due to the Raft protocol backing the ordering service. Different organizations may have orderer nodes in the same channel, meaning that no one organization would have to depend on a single centralized host for transaction processing and consensus, and would also be able to handle one of the other organizations' hosts crashing. However, the Raft ordering service is not a byzantine fault tolerant system, meaning malicious orderer nodes may affect the transaction ordering. This could be mitigated by giving each user a single orderer for their own channel.

The modification would center around the process for joining the network. Whenever a new user joins the network, they will be presented with the choice between downloading the network application and API for usage on a personal server (for advanced users), and a choice between service providers who can host your API and corresponding HLF gateway peer. The data is stored in the same location as the rest of the components that constitute the new user's membership in the network, which isolates it from unauthorized access as well as giving the user complete control over the data. The cost of participating in the network would therefore be the eventual hosting fee you would have to pay to the external service provider. And this way, the cost of participation in the network is completely transparent in the way that you pay for the resources needed to host the application, and not with your personal data.

Another benefit of decentralizing the application is the potential performance gain. Performance of the system would depend on the popularity of any single user, and the processing power they have delegated to processing transactions. For the most part, this would result in stable performance.

An important factor when creating any software product is the value proposition: is there any money to be made off the product? For income, the developers of the network could offer a Software-as-a-service package where users can pay the network for hosting the peer associated with their account. Payment could vary depending on the resources needed for the user that is signing up; a high profile celebrity signing up will likely require more bandwidth and performance than a regular citizen.

VIII. Conclusion

Personal data is valuable. It is easy to claim that not opting out of having your private information recorded does not affect you because you cannot feel the physical effects. However, if data collection to this degree is allowed to run as wild as it currently is, the large companies stand free to use the information to further their own agenda. One needs to look no further than the Cambridge Analytica scandal to see the effects of this. All humans have an inherent right to privacy, and the amount of data that is collected is tantamount to constant surveillance. All this data in the wrong hands can result in the loss of personal freedom.

The finished application does not fully exploit the capabilities of Hyperledger Fabric. The mainstay of hyperledger fabric is its ability to adapt to different use-cases, and only using the test network is severely limiting, both performance- and security-wise. However, our project shows that it is possible to create and maintain a social network using Fabric. Personal data is protected from users that should not have it, and all operations on data is recorded in the ledger, so you can audit how your data has been used by others.

Despite us making a case for using blockchain in social media applications, there must be room for the consideration that blockchain might not be the best solution to the problem. While knowing where posts are stored, as well as being able to track where posts are traveling through the network can be of interest to the owner of the posts, the solution might be overkill for a simple social network, and instead be more applicable in a business context. Most people using social media aren't too concerned with who reads their posts, hence the massive user base of Twitter and Facebook. In a business context, the data you share with other members might be a lot more sensitive, and users as well as the companies they act on behalf of may have a stronger interest in maintaining a log of how the data shared between companies is used. The biggest problem with social media today is not being unable to track who shares the posts you share publicly with friends, but rather how the companies that provide the social network processes all the other data that you indirectly give away through actions such as scrolling, liking and sharing posts. This is a problem that implementing blockchain alone does not solve. Though some claim it could be solved with simply open-sourcing the code base associated with the social media, simply showing what information you are collecting is not the same as not collecting it.

IX. Acknowledgements

Thanks to Chunming Rong and Jiahui Geng for proposing the task, and for guidance and supervision during its execution.

References

[1] A. Ringen and L. A. Jaatun, "Just Us: A Blockchain-based Privacy-friendly Social Network," UiS BSc Thesis, 2022.
[2] "What are smart contracts on blockchain?" [Online]. Available: https://www.ibm.com/topics/smart-contracts
[3] "Smart contracts and chaincode," https://hyperledger-fabric.readthedocs.io/en/latest/smartcontract/smartcontract.html.
[4] "Hyperledger fabric - a brief history." [Online]. Available: https://www.linkedin.com/pulse/hyperledger-fabric-brief-history-binh-nguyen
[5] "Endorsement policies," https://hyperledger-fabric.readthedocs.io/en/latest/endorsement-policies.html.
[6] "Channel configuration (configtx)," https://hyperledger-fabric.readthedocs.io/en/latest/configtx.html.
[7] "The ordering service," https://hyperledger-fabric.readthedocs.io/en/release-2.2/orderer/ordering_service.html.
[8] A. Barger, Y. Manevich, H. Meir, and Y. Tock, "A byzantine fault-tolerant consensus library for hyperledger fabric," in *2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2021, pp. 1–9.
[9] "Identity," https://hyperledger-fabric.readthedocs.io/en/latest/identity/identity.html.
[10] "Steem: Powering communities and opportunities." [Online]. Available: https://steem.com/
[11] "Somee: Social media redefined for privacy, end user control and monetization," March 2022. [Online]. Available: https://somee.social/
[12] "Sapien: Social network for privacy and intependence." [Online]. Available: https://www.sapien.network/
[13] S. Cunningham, "Somee's last effort to take your money," February 2022. [Online]. Available: https://www.publish0x.com/at-scottcbusiness/somee-s-last-effort-to-take-your-money-xyyrlgg
[14] Sapien, "Sapien white paper," https://coinpare.io/whitepaper/sapien-wallet.pdf, 2020.
[15] Steem, "Steem white paper," https://steem.com/steem-whitepaper.pdf, 2018.
[16] J. Goodwin, "Social media shouldn't run on personal data," https://blog.sapien.network/social-media-shouldnt-run-on-personal-data-44fcd2fc29ad, 2020.
[17] A. Chakravorty and C. Rong, "Ushare: user controlled social media based on blockchain," in *Proceedings of the 11th international conference on ubiquitous information management and communication*, 2017, pp. 1–6.
[18] "React native · learn once, write anywhere," January 2022. [Online]. Available: https://reactnative.dev/
[19] "Meta," January 2022. [Online]. Available: https://about.facebook.com/meta//
[20] "Asyncstorage · react native," March 2022. [Online]. Available: https://reactnative.dev/docs/asyncstorage
[21] "Using the effect hook," March 2022. [Online]. Available: https://reactjs.org/docs/hooks-effect.html
[22] "Flatlist · react native," February 2022. [Online]. Available: https://reactnative.dev/docs/flatlist
[23] "Hyperledger fabric go contract api," https://pkg.go.dev/github.com/hyperledger/fabric-contract-api-go.
[24] "gorilla/mux," https://github.com/gorilla/mux.
[25] "Code editing. redefined," February 2022. [Online]. Available: https://code.visualstudio.com/
[26] "Gorilla mux," February 2022. [Online]. Available: https://pkg.go.dev/github.com/gorilla/mux
[27] "Hyperledger fabric: Using couchdb," March 2022. [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-2.2/couchdb_tutorial.html
[28] M. G. Jaatun, P. H. Haro, and C. Frøystad, "Five things you should not use blockchain for," in *2020 IEEE Cloud Summit*, 2020, pp. 167–169.
[29] "Elon Musk to Acquire Twitter," https://www.prnewswire.com/news-releases/elon-musk-to-acquire-twitter-301532245.html, April 2022.