College of Engineering

COMP 491 – Computer Engineering Design Project

Final Report

Infrastructure to Migrate to Your Own Avalanche Subnet as a Dapp

Metin Arda Oral - 69205
Mehmet Enes Erciyes - 68906
Ziya İçöz - 69022
Tolgay Dülger - 68881

Project Advisor(s)
Assoc. Prof. Alptekin Küpçü

Spring 2023

Abstract

This project addresses the common challenges that decentralized applications (dApps) encounter during their migration to their own blockchain. Focusing particularly on the Subnet architecture of the Avalanche Network, we have considered scenarios where dApps migrate from an EVM-based blockchain to their own EVM-based blockchain. We have successfully established a new blockchain locally by utilizing Avalanche's Subnet technology. We ran the local subnet on a VPS (virtual private server) and made it available for everyone. To allow bridging the existing token to the new blockchain. We have deployed a bridge between the existing and new blockchains by deploying smart contracts on both chains and running a relayer application. In addition, we have implemented a web-based user interface for users to easily bridge their tokens. Additionally, we have created a simplified blockchain explorer for the new blockchain. First, we implemented a blockchain indexer application. Which runs continuously to process each block and save related data to a database. We have created a RestAPI that server the data created by the indexer application to our web-based block explorer. This explorer displays transactions on the blockchain in real time. Finally, we have implemented a social media application for Android and iOS that utilizes the bridged tokens on the new blockchain. In essence, this project has successfully overcome the main challenges of dApps migrating to their own blockchains while enhancing user experience and maintaining real-time visibility of transactions.

1. Introduction

In the current world of blockchains, multiple applications coexist on the same blockchain and because of the concept of gas, users of an application could be faced with high gas fees unrelated to the application they are interacting with. Therefore, gas price is not isolated for each application. A good example of this situation would be the NFT mint of Otherdeeds by Yuga Labs (creators of BAYC) [1]. Because of the high demand for this mint, gas prices have arisen unacceptably on the Ethereum network at that time, which caused users that were interacting with other applications to either pay expensive fees or not be able to interact because it was not feasible with given gas prices at that point. This example opened up a discussion on Yuga Labs migrating to their own blockchain. The argument is as follows, high demand applications such as Yuga Labs' NFT mints should be on an application-specific blockchain so that they would have higher output on their own and they would not clog up the Ethereum network and decrease the efficiency of other applications on the Ethereum network.

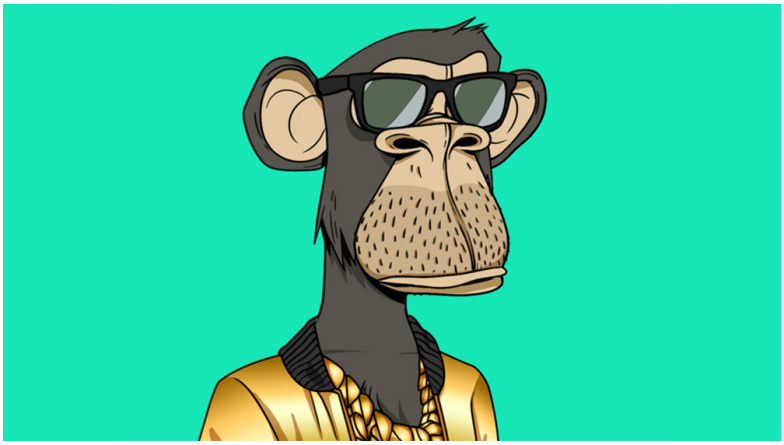

Figure 1. Famous "Bored Ape" NFT from Yuga Labs

Another example would be the Crabada, which used to live on Avalanche C-chain and then migrated to their own Subnet. Crabada used to be responsible for around 16.19% of the gas used on Avalanche C-chain [2], therefore, raising the gas prices. To scale better, they decided to migrate to their own Subnet and it allowed Avalanche C-chain to be more scalable.

The examples above are related to the throughput of the blockchain but there could be other reasons why you might want to use your own blockchain. As it could be seen from the Avalanche Subnet's website [3], if you want to use your own token as the native token of the blockchain, you might have to comply with regulations and have to have some control over validators, contract deployers, or users that interact with your applications. In that case, you would want to migrate to your own blockchain which allows you to have some access control that is not possible with public blockchains such as Ethereum or Avalanche C-chain. An example of such a case is Intain MARKETS [4].

As demonstrated above, there are convincing reasons why migrating to one's own blockchain is an important process to solve, and several companies are already working to make this migration. We have seen projects trying to migrate to their own blockchains, and we believe this migration will become more common day by day.

The first problem is bridging existing assets to their new blockchain. Crabada solved this problem by collaborating with other projects that work on bridging [5], they have partnered with LayerZero labs and Celer cBridge. Another project that migrated to their own subnet is Defi Kingdom (DFK), they have their own chain right now and partnered up with Synapse to provide a bridging solution [6]. Another project that is still in development is Kaira Network, this project supports bridge functionality on their own [7]. From the above examples, we can see that new Subnet projects have this need to bridge.

The second problem is the need for a block explorer. All of the projects above need some kind of a block explorer because they have their own blockchain. They again get support from different projects. For example, Crabada and DFK collaborate with Ava Labs on their block explorer [8], whereas projects like Kaira collaborate with AvaScan [9].

In this project, we aimed to solve these two problems with our own solutions, providing the infrastructure for a hypothetical social media dApp that uses the tokens in a blockchain as an authentication mechanism.

2. System Design

First of all, to solve the problems of a hypothetical company by providing this interface, we had to mock a dApp (decentralized application) that was trying to migrate to their own blockchain to solve their problems. Therefore, we had to create a dApp and a blockchain.

2.1. Social Media dApp Mock

In our hypothetical scenario, ZATE Social application is a social-media app for the users of ZATE blockchain network. This exclusive community is only open to the users who have ZATE tokens in their wallets. However, due to the overwhelming interest in the application, ZATE Co. is evaluating to migrate to their own blockchain network. This section is about the implementation of this mock social media application.

We have developed a social-media mobile application that leverages the innate token balance of each user within the blockchain infrastructure for authentication. To ensure a seamless experience, we opted for Flutter, a powerful cross-platform framework that allows us to cater to both iOS and Android users effortlessly. By harnessing the capabilities of Flutter, we can deliver a cohesive and visually stunning interface that enhances user engagement and satisfaction.

In order to establish a robust and scalable backend system, we integrated Firebase, a widely acclaimed platform renowned for its versatility and reliability. By harnessing the power of Firebase, we ensure efficient data management and seamless cloud integration.

While our overarching project comprises multiple components, our social media app takes center stage, serving as a focal point for showcasing all the work put into providing the infrastructure for migrating to a new chain.

In Figure 2, you can see the working authentication flow of the mobile application. The most critical part here is the implementation of a service that connects to MetaMask. MetaMask

is a popular cryptocurrency wallet and browser extension that allows users to manage their digital assets. By connecting to this wallet, we can assess if the user has the tokens to access the app. After the authentication is complete, user is required to create an account and then they are invited to freely roam the world of Zate Social. The application is a very basic social media application that allows users to follow each other, post pictures and like them. We have designed our application to be visually appealing.

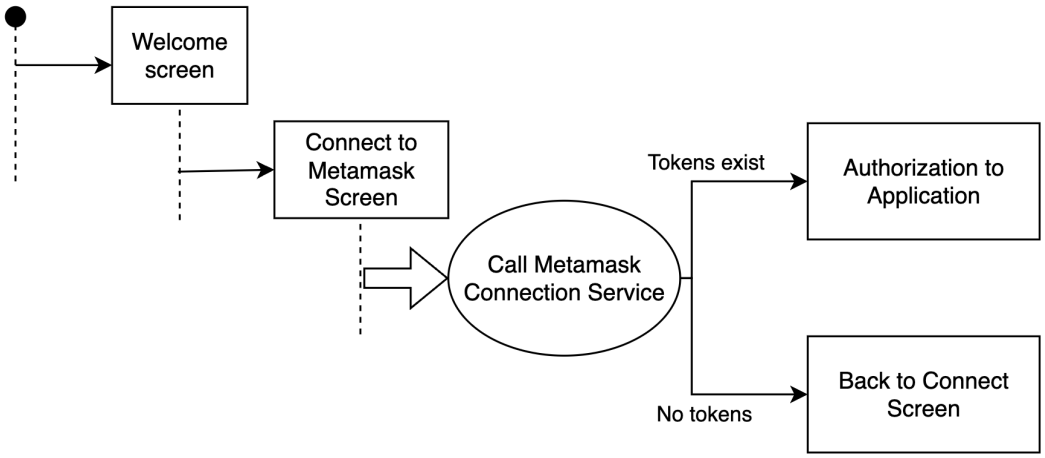

Figure 2. Authentication flow of the mobile application

The user registration and backend of the application uses Firebase. In Figure 3, you can see the database schema of the application.

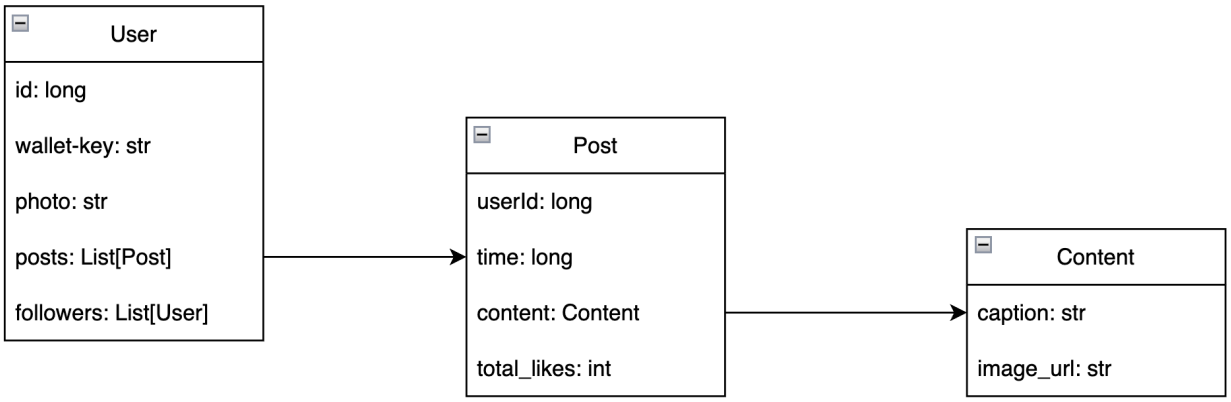

Figure 3. Firebase database schema

Lastly, Figure 4 and Figure 5 shows screens from our application. Figure 4 displays the "Connect to Metamask" screen and Figure 5 displays our basic social media application feed with our mascot "Fluffy".

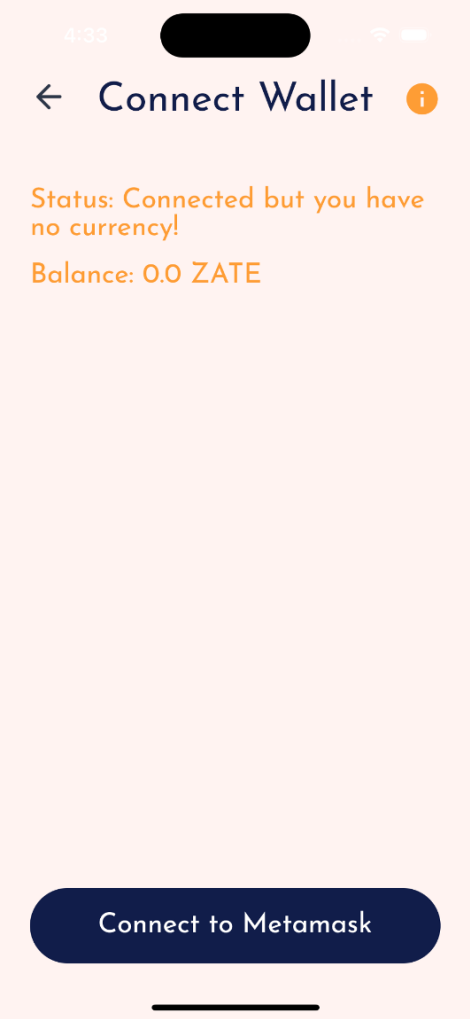

Figure 4. Connect to Metamask screen

Figure 5. Social Media Sample Feed

2.2. Subnet

Figure 6. Logo of the blockchain network platform we used

When it came to creating a blockchain, we went with `avalanche-cli` as a tool. We wanted to use Avalanche Network shown in Figure 6 in general to make use of stateful precompiles, especially NativeMinter precompile, which allows us to mint native tokens of the blockchain. Therefore, we have created a blockchain using avalanche-cli and enabled NativeMinter precompile. Since we wanted everybody to be able to interact with our blockchain, we ran this blockchain on a VPS and configured the RPC to allow accepting requests from any IP address.

We used Linode free credits to rent a VPS and run the system.

2.3. Block Explorer

We have achieved to mock a dApp that was trying to migrate to their own blockchain. The next step was to solve their common problems. First, we went with solving the common issue of lacking a block explorer. Blockchains do not store the data in an understandable format for an average user. In addition, it does not store the data in a way that frontend applications are used to. Therefore, we had to process what was happening on the blockchain and serve it in a way that front-end applications are used to. To solve this problem, we have decided to come up with an application, whose whole purpose is to connect to the blockchain, subscribe to new blocks, process those blocks to get any related information, and save that data in a database. We have chosen to use Typescript with Node Js runtime environment and MongoDB as our database to write this blockchain indexer. Our application would get transactions of a block and save transactions and blocks to the database. A sample process is illustrated in Figure 7.

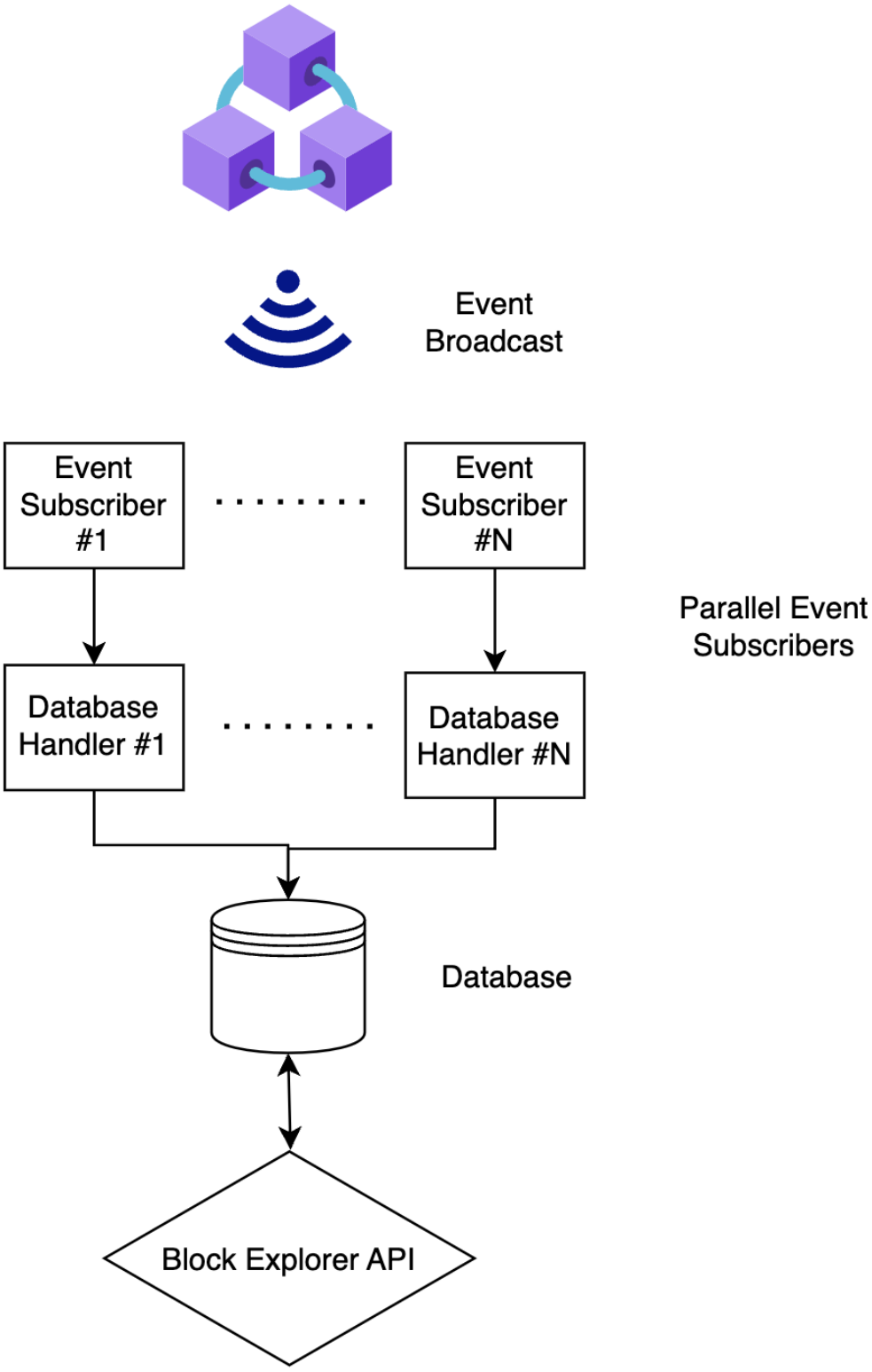

Figure 7. System design of the Block Explorer

This was the data creation process, we had to write a RestAPI for our frontend to consume. Therefore, we have written a RestAPI using Typescript with Node Js runtime environment that connects to the same MongoDB as our indexer. This Rest API had a couple of endpoints to allow getting the latest blocks and transactions, getting specific blocks or transactions, getting the native token balance of an address, and getting transactions related to an address. After all of this work, we finally had the data we wanted in the format we wanted. Finally, we have written a web-based interface for users to consume this data. We have used Next Js with React to build up our web user interface. As a result, we were able to show the user real-time blocks and transactions happening in our blockchain. An image from this web interface is shown in Figure 8.

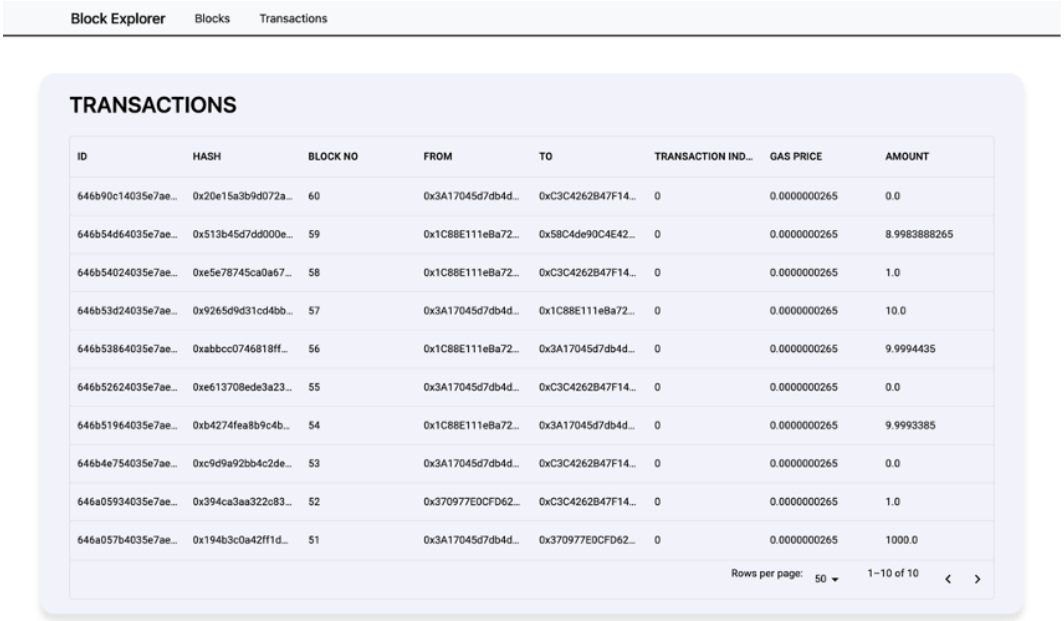

Figure 8. Screenshot of the transactions on our network from our Block Explorer

2.4. Bridge

The second common issue was to migrate their existing token to their new blockchain. Since blockchains are isolated applications they are not able to be in contact with other blockchains, it is not an easy task to use a token that is on another blockchain. We had to come up with an idea to fill this gap between those two blockchains. We thought about the interactions we can do with a blockchain. First, you can send a transaction to the blockchain to change the state, secondly, you can observe the state changes of a blockchain. Building upon those two

simple ideas. We have come up with an architecture where we had smart contracts on both chains and an external application that listens to both blockchains' state updates and sends transactions accordingly to the other blockchain. How it works is as follows, a user would send a transaction to lock/burn their token, this could happen on either of the blockchains. This transaction would cause a state change and inside the contract, we would add an event to let external applications know about the state change. External applications would observe this and build up the adequate transaction to send to the other blockchain. In our case, if it sees a token lock, it would send a transaction to mint tokens on the other blockchain. As a result, this relayer application allowed us to talk in between blockchains. This was the high-level overview of our bridge architecture. When it came to implementation, we deployed an ERC20 token to the Avalanche Fuji network to mock an existing token. Afterward, we have written smart contracts in Solidity, using Hardhat as our development tool. In addition, we have written our relayer application in typescript. This script was listening for events on both bridge contracts that are deployed to both blockchains. When it observes an event, it would build up the adequate transaction and add the transaction to the transaction queue it has in memory. Every 5 seconds, it would send a transaction. To be able to easily test this process, we have written a couple of scripts to interact with those contracts and test the functionality of our bridge. After making sure that our bridge was working as expected, we have developed a web-based user interface, with Next Js and React, for users to bridge their own tokens. Figure 9 shows a screenshot of our bridge website.

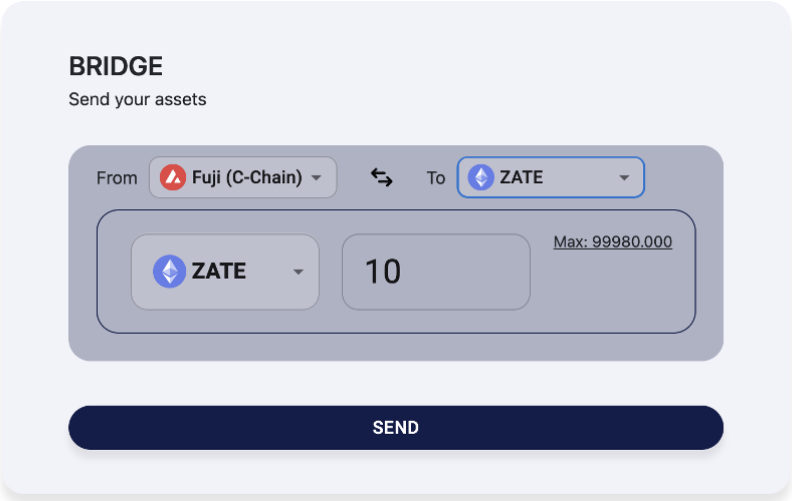

Figure 9. Bridge website screenshot

3. Analysis and Results

After implementing our design, we had 8 components that work in harmony to achieve our goal and successfully demo our use case. We created:

1. A custom blockchain on our VPS
2. A Flutter mobile app for demo
3. A blockchain indexer
4. An indexer api
5. A block explorer web interface
6. Bridge smart contract
7. Bridge relayer
8. Bridge web interface.

Our design worked great considering that all of these pieces are connected, it was not an easy task to make sure that they were correctly separated yet worked together. Figure 10 shows how all of these components fit together in our project.

We are happy with our design choice of using Avalanche to create our blockchain, but we have struggled while implementing it. We had problems running the local blockchain on our VPS, we later found out that the hardware specifications of our VPS were not good enough. We had 8 GB of RAM and then switched to 16 GB of RAM. After being able to run our local blockchain on the VPS we faced another issue, which was the fact that we could not use RPC to communicate with the local blockchain. Turns out, by default, it is closed to outer IPs therefore, we had to find and update the configuration file for the node. These were the two main challenges we faced that we did not expect while planning our project.

We are satisfied with our decisions on the design of our mobile application. It successfully allows users to connect their Metamask account and check their balance. Those two features allowed us to successfully mock a dApp. We could have done more on the mobile application part by making use of smart contracts to implement features like liking a post and adding a comment. But we had less time than we expected because of the unexpected problems

we have faced during our implementation. Considering this application was to mock a dApp, we were successful. It is just that we could have done more.

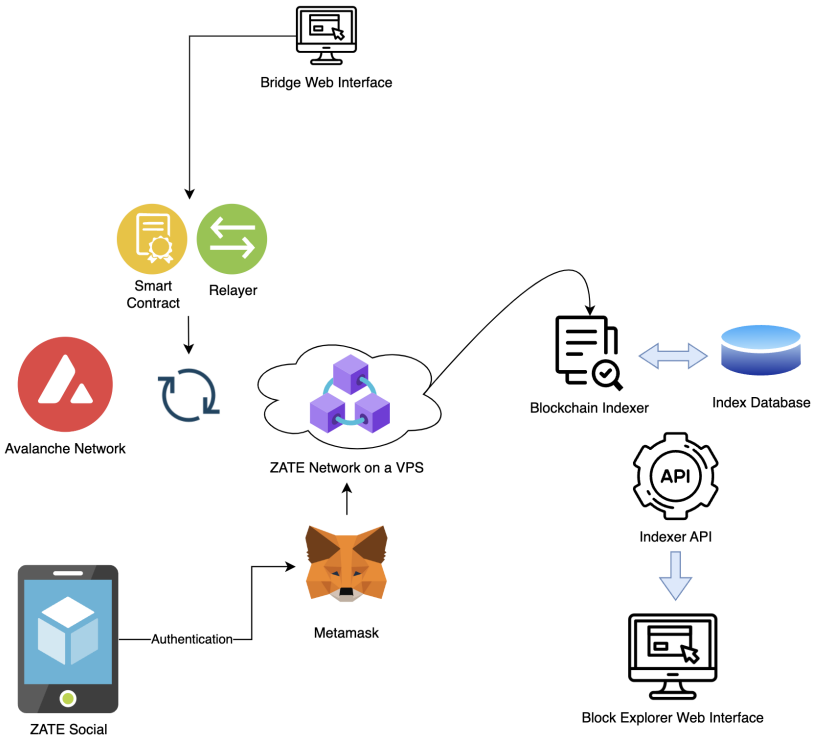

Figure 10. General system design of the project

We are satisfied with our block explorer application design. It allows users to see blocks and transactions happening in our blockchain in real time. It has all the essential information. It certainly successfully helped us with our bridging process by showing the transaction that is sent by the relayer. As a result, users were able to understand the data on the blockchain easily.

We are feeling great about our bridge design. The decision to use Avalanche Consensus on both blockchains, allowed us to not wait for any amount of blocks for confirmations, we were able to initiate the bridging process as soon as the transaction was included in a block. Other than that, we are happy with our solution on how to connect different blockchains and think that this idea could be extended to support multiple tokens. In our case, this was not needed therefore, we did not implement such a thing, but it could be a cool feature.

We are happy with how our software solves the common problems of dApps migrating to their own blockchain. But, after finalizing the project, we developed the idea of creating a CLI tool for developers to quickly deploy our software with their configurations. This CLI tool would be the glue to hold all of our software together and would be the single control point for developers to handle their migration infrastructure. It would abstract the unnecessary complexity for other developers and allow them to easily bootstrap the infrastructure. This project can be a great follow-up future work to our project.

4. Conclusion

Our infrastructure solutions aim to provide a Web3 company with easy tools to migrate to their own blockchain network. To this end, we created a custom blockchain network in the cloud called the ZATE network. Then, we created a sample social media app that requires users to have some ZATE token in their wallets to access their exclusive network. Then, we designed a system with multiple units to provide our users with an interface to bridge their tokens on the old network to their new one. Furthermore, we implemented a block explorer interface to allow users to check if their transactions occur correctly. The output of our project can be utilized to provide a blueprint for such companies.

APPENDIX

Source code of the project:

Indexer: https://github.com/zicoz18/COMP491-Indexer
Indexer API: https://github.com/zicoz18/COMP491-Indexer-API
Bridge Contracts and Relayer: https://github.com/zicoz18/COMP491-Bridge
Subnet Config File: https://github.com/zicoz18/COMP491-subnet
Bridge Web App: https://github.com/dulgertolgay/COMP491-Bridge
Block Explorer Web App: https://github.com/dulgertolgay/COMP491-block-explorer
ZATE Social Mobile App: https://github.com/ardaqwe35/COMP491_mobile_frontend