Slovak University of Technology in Bratislava
Faculty of informatics and information technologies

Reg. No.: FIIT-100241-102906

Igor Ďurica

Internet of Value and DeFi in Solana

Bachelor's thesis - Appendices

Study programme: Informatics

Study field: Computer Science

Training workplace: Institute of Computer Engineering and Applied Informatics

Thesis supervisor: Ing. Kristián Košťál, PhD.

May 2023

A. Project task schedule

A.1 Winter semester

1st week	Surface level research of the Solana de-fi ecosystem
2nd week	Deeper research into potential topics
3rd week	Final selection of a topic, deeper research into selected topic
4th week	Introduction to blockchain technology, common de-fi solutions
5th week	Brief overview of the Solana blockchain, proof of history
6th week	Analysis of existing solutions with a similar focus to that of the selected topic
7th week	Analysis of shortcomings of existing solutions
8th week	Polishing and expanding of previous sections of the thesis
9th week	Overview of own proposed solution
10th week	Initial design of own solution
11th week	Deeper dive into the solution's design
12th week	Proposed metrics and methods for testing the solution
13th week	Foundation for the implementation

A.2 Winter semester work evaluation

As of the writing of this section, the majority of the work outlined in the winter semester schedule has been completed. We did unfortunately underestimate the amount of time required to fully cover all existing solutions as well as their shortcomings in the analysis section. Both parts of this section ended up taking roughly a week longer than initially anticipated. The analysis therefore was not complete until the 11th week. As a result, the sections relating to testing and a development of a basic foundation for the implementation have been pushed into the summer semester instead. The schedule for the remaining work seems very achievable despite these delays however, and the risk of the thesis not being complete on time seems low.

A.3 Summer semester

1st week	Surface-level solution implementation design
2nd week	Detailed solution implementation design
3rd week	Oracle smart contract implementation
4th week	Oracle node implementation
5th week	Web application implementation
6th week	Review and consolidation of the implementation
7th week	Implementation analysis and technical documentation
8th week	Implementation testing
9th week	Report and assessment of testing results
10th week	Final review and adjustments to implementation
11th week	Conclusion and future work sections
12th week	Resumé section
13th week	Retrospection and final document adjustments

A.4 Summer semester work evaluation

We unfortunately experienced multiple delays and setbacks during the summer semester period, all of which resulted in falling behind on the schedule outlined above. Numerous parts of the implementation were not completed until the 13th week. This was primarily caused by technical difficulties mostly stemming from the fairly limited documentation available for Solana and the anchor framework. Our original estimates for the complexity of the implementation also proved to have been rather optimistic. More time and effort was ultimately needed to complete the implementation than what had originally been estimated. Additionally there were other setbacks caused by events outside of our control such as an update to

the phantom wallet, which rendered some parts of the wallet adapter used in our application incompatible, further complicating the development process.

B. Technical documentation

B.1 Requirements

General requirements:

- rust - https://www.rust-lang.org/tools/install
- node.js & npm - https://nodejs.org/en
- yarn - https://classic.yarnpkg.com/lang/en/docs/install/#windows-stable
- solana - https://docs.solana.com/cli/install-solana-cli-tools
- anchor - https://www.anchor-lang.com/docs/installation
- phantom - https://phantom.app/download

Running Solana programs locally requires a UNIX-based operating system, if you wish to use Windows, you will require the following:

- wsl - https://learn.microsoft.com/en-us/windows/wsl/install
- vscode - https://code.visualstudio.com/download
- wsl for vscode - https://marketplace.visualstudio.com/items?itemName=ms-

vscode-remote.remote-wsl

B.2 Setup

Notes:

- All folder paths mentioned will be relative to the implementation root directory (BP_IgorDurica)
- Make sure all dependencies listed under General requirements are installed

B.2.1 Solana configuration

- **(WSL ONLY)** Open a new instance of vscode as administrator and connect to your wsl instance via the vscode extension (icon should be visible in the bottom left corner of the vscode client)
- **(WSL ONLY)** Open a terminal inside of your vscode instance (your local solana validator will not launch inside of a regular wsl terminal)
- **(NON-WSL ONLY)** Open any terminal of your choice
- Enter "solana config set –url localhost"
- Enter "solana-keygen new" - this will generate a .json file containing the keypair to your solana wallet saved as an array of integers (first 32 integers represent the private key, last 32 integers represent the public key)
- Enter "solana-test-validator"
- Open "/bp-smart-contract" in a terminal
- **(WSL ONLY)** Enter "wsl"
- Enter "anchor build"

- Enter "anchor deploy"
- Enter "solana address -k target/deploy/solana_twitter-keypair.json"
- Copy the id from the terminal
- Open "/bp-smart-contract in a terminal/Anchor.toml"
- Change the value of "oracle_smart_contract" to the string you copied earlier
- Open "/bp-smart-contract/programs/oracle-smart-contract/src/lib.rs"
- Change the input inside of the declare_id! macro to the string you copied earlier
- Enter "anchor build" into the terminal
- Enter "anchror deploy" into the terminal
- Solana initialisation is complete and your contract is deployed

B.2.2 Web app initialisation

- Open the web browser which has your phantom wallet extension installed
- Open the phantom wallet extension window
- Create a new phantom wallet account
- Enable testnet mode inside of your phantom wallet
- Switch the solana network of your phantom wallet to "Solana testnet"
- Copy the address of your phantom wallet
- Open a terminal window
- **(WSL ONLY)** Enter "wsl"

- Enter "solana airdrop 100 {your wallet address}"
- Open "/bp-frontend" in a terminal
- Enter "npm install"
- Enter "npm run serve"
- Navigate to "http://localhost:8080/#/" in your browser (make sure your phantom wallet extension is installed in the same browser)
- Press the initialise button - this will trigger a total of 4 transactions, make sure to confirm each transaction, this may take a while - note: the phantom wallet chrome extension recently received an update however the wallet adapter used by the web application is still configured to work with the previous version, various issues and errors may arise while using the wallet
- Copy the state address from the console
- The web application setup is complete, you can now create subscriptions
- Note that if you refresh the page, you will have to initialise it again from the home page

B.2.3 Oracle network initialisation

- Open "/bp-oracle-network/oracle-server" in a terminal
- Enter "npm install"
- Open "/bp-oracle-network/oracle-server/src/index.ts" change the value of stateAddress to the address you copied from the console at the end of the web app initialisation
- Open "/bp-smart-contract/C UsersIgor.config/solana/id.json" and copy the

contents of the file - note that the name of this directory may be different for you

- Open "/bp-oracle-network/id.json", delete the contents of the file and paste the copied contents of the other id.json file
- Open "/bp-oracle-network/oracle-server" in a terminal
- Enter "npm run build"
- Enter "npm run serve"
- Open "/bp-oracle-network/oracle-node" in a terminal
- Enter "npm install"
- Enter "npm run build"
- Enter "npm run serve"
- Your oracle network should now be up and running, if you have any existing subscriptions and the web application has been initialised properly, you will be able to see the contents of API responses in the oracle server terminal

B.3 Implementation details

B.3.1 Oracle smart contract

All application logic can be found inside of "/bp-smart-contract/programs/oracle-smart-contract/src"

"lib.rs" contains all instructions. Instructions are effectively just methods that the outside world can use to interact with the contract. They are passed a Context object which contains the accounts that the instructions is supposed to read or

write to. Since Solana programs are stateless, these accounts are how we store data. Most relevant application state is stored in our State account.

"state.rs" contains the details of the state account. This is where the proof of stake logic is located, among other things. The select_leader and generate_seed functions are responsible for the proof of stake mechanism. The seed used is taken from a solana system variable containing recent slot hashes.

"context.rs" contains the implementations for individual context objects for our instructions which dictate what accounts must be supplied to an instruction when it is called. When creating a new account, we must annotate it with "init" and specify an exact size. We must also provide a system_program attribute.

B.3.2 Web application

All application logic besides certain configurations can be found in "/bp-frontend/src/components/" and "/bp-frontend/src/composables".

"useWorkspace.ts" is where we define a connection to our smart contract which can then be used by our components. It also contains references to other useful variables.

"/bp-frontend/src/components/" contains individual component files, all of which are fairly straightforward

B.3.3 Oracle network

All application logic besides certain configurations can be found in "/bp-oracle-network/oracle-node/src/" and "/bp-oracle-network/oracle-server/src/".

The oracle server in this case is a socket server and each oracle node is an individual socket. Both of these have a connection to our smart contract which they use to

fetch data. In the case of the oracle server, it also reports data. Both sides have emit handlers for various messages which they use to communicate amongh each other.

Typical program flow goes as follows:

- The server is initiated.
- At least one socket (oracle) connects.
- A new round starts.
- All oracle fetch subscriptions and then call the endpoint for each subscription.
- Each oracle reports all collected data.
- The oracle server sends reported data to the leader who saves all of it.
- The leader generates a hash for each report JSON.
- Hashes are stored in a hash map and a record is kept of which hash was seen the most times.
- The most popular hash is chosen as the final report and submitted to the server.
- The server broadcasts the report to each oracle.
- Socket generate a hash of their own collected report and the chosen report and compare them. If the hashes match, the oracle votes in favor of the report, otherwise it votes against it.
- Upon receiving votes from all participants, the server checks if at least two thirds voted in favor of the report.
- If the report received enough votes, data is reported to individual subscrip-

tion accounts.

- Regardless of the outcome, the end of the round is reported to the smart contract, along with information of whether the report was accepted or not.
- A new round begins.

C. Contents of the digital medium

Registration number of the thesis in the information system: FIIT-100241-102906

Contents of the digital medium (ZIP archive):

Folder	Contents
/bp-frontend	web application implementation, configs
/public	placeholder files
/src	application setup files and logic, other folders
/assets	images
/components	component files for individual pages
/composables	files for global utility functions
/bp-oracle-network	oracle network implementation
/oracle-node	oracle node implementation folder, configs
/src	oracle node implementation files
/oracle-server	oracle server implementation folder, configs
/src	oracle server implementation files
/bp-smart-contract	oracle smart contract implementation, configs
/C UsersIgor.config	wallet store folder
/solana	solana wallet private key as json
/migrations	deployment script
/programs	contract implementation folder
/oracle-smart-contract	source code folder, configs
/src	smart contract source codes
/tests	unit test files

Name of the submitted archive: BP_IgorDurica.zip.