Reframery Capstone Project - DANC Coin
CS 4ZP6A/B
Dated 21/11/2020

Group Members:
Daniel Wu
Charles Zhang
Aly Shah Imtiaz
Nimalan Chandrasekara

1. Anticipated Changes

A) Data Representation

- *Type of User Information* - may vary based on how much intel client wants to store.
- *API for Frontend to Backend* – type of variables from frontend to backend will see changes as more and more features are implemented to both ends. Type of usability is currently expanding, and hence more information will be required to be transferred from frontend to backend and vice versa.

B) Behavior:

- *Transaction:* users will have the ability to make transactions between accounts and view previous history. Expansion of what users can also do will be included as the client proposes other ideas.
- *Transfer DANC Coin* – bottleneck of the system as worst case is 5 minutes per transaction. Users will prefer not to wait this long, currently exploring methods to reduce this on the backend or reduce wait time for users on the front end by accepting transactions first and then processing second.

C) Hardware Specifics:

- *None* - our software will be completely online.

D) Configuration

- *Currency Operations* – currently include having the ability to make a transaction between accounts and view previous transactions. Changes may include DANC currency to physical cash, transfer between multiple accounts at the same time, etc.
- *Fault avoidance* – currently practicing standard programming practices such as peer reviews to avoid bugs in codebase. Anticipated changes seen here with more interaction with clients and their backend team to further prevent bugs and smoothen the system.

E) Security Requirements:

- *User login Authentication & Authorization* – will be prevented via JWT (JSON web tokens). Because users will login to their account using their email and password, stronger security measures will be put in place. Currently, basic security measures are in place to help prevent intruders. JWT will allow users to log in once throughout the whole session.

- *51% Ethereum Network Attack* – prevention will be from the cost of mining powers. This essentially causes double spending when a malicious actor spends their DANC coin balance twice. Highly unlikely for this to occur, however given the importance of the currency to every single customer, we anticipate stronger measures potentially being put in place as we move down the road with the client.

2. Unlikely Changes

A) Data Representation:

- *Type of Currency:* will remain virtual and based on specific local community's value.

B) Behavior:

- *Log in Method* - will remain the same with users inputting email and password.

C) Hardware Specifics:

- *Client accessibility* – will use either mobile phones or desktop to access apps. No physical bank will exist.

D) Configuration

- *Use of Ethereum* **–** no changes here as entire software is currently built off of Ethereum blockchain as it has best use cases for client's purpose.
- *PostgreSQL database* – user information currently stored in a PostgreSQL database, no anticipated changes for storing this information in another format.
- *Scalability* – backend will be hosted on AWS, no changes seen here. AWS will be able to handle all current scalability concerns with the app such as automatic load balancing.
- *Frontend to Backend API Connection*– connection between frontend and backend will have no changes as one depends on the other to show use information and process that user information.
- *Backend to Ethereum Connection* – no changes will be made here as connection is required to complete transactions.

E) Security Requirements

- *PostgreSQL database protection* – will be prevented via ORM, this will help ensure that the webapp risk will be mitigated.
- *Backend database* – will be hosted on AWS, no changes required for security measures here as those measures are already in place by AWS.

3. Concepts

Blockchain concept and issues:

- *Block confirmation concept:*
    - Run-time: The ethereum blockchain uses the concept of blocks of transactions. Transactions placed in blocks are not considered “confirmed transactions” until many blocks later to reduce the risk of double spending. This is an inherent risk because of the decentralized nature of blockchains.

- *Transaction time issue:*
    - Design-time: Users should see a transaction go through right away on the frontend while the blockchain end processes the transaction. This is not a simple design problem to solve however because transactions may not always go through on the blockchain side. Many things can cause this such as double spending and network congestion. Transactions would need to be resent if it fails to process. We will need to come up with a unique solution to solve this problem.

Security concepts:

- *Authorization, Login to system:*
    - Design-time: process in place for when a user is prompted to log in to application (username: email address, and password).
    - Run-time: ensuring each login is validated on its own and not on previous history of logins. Ex. a user is required to provide credentials for every instance of their access to the application.

- *User database:*
    - Design-time: ensuring user data is captured and stored in a safe environment where only those authorized to access can. Other users should not be able to see another user’s information.

- *Ethereum Smart Contract:*
    - Design-time: proprietary codebase that allows the application designed to be unique. In order to ensure the algorithms are encapsulated, security measures will be required as this will not be open-source.

REST API concept:

- Run-time: A way to facilitate communication between the frontend and backend application.

Database concept:

- Design-time: Store user information with the appropriate db schema

Cloud hosting:

- Design-time: Host our backend service and db on a cloud platform (AWS)

4. Module Secrets

Module	**Secrets**
Ethereum API library	- Communication protocol between blockchain and backend - Ethereum blockchain: node to node interaction - Ethereum blockchain: miner (worker) interaction - Transaction queuing
Ethereum smart contract	- Method of transaction between wallets - Method of token generation - Burning tokens
User database	- Row ordering - Data schema - Keys - Indices
API for bridge between frontend and backend	- Http protocol
API for bridge between db and blockchain	- Http protocol
Validation	- Method of validation (anonymous function to make validation generic) - Http code to return to frontend team - Message to show to frontend team

5. Model Interface

User database:

Table Name	Purpose	Inputs	Outputs
Users	Stores user email, password (hashed + salted) name, phone number, address.	N/A, SQL queries will be run.	N/A, SQL queries will be run.
Transactions	Stores each transaction, and whether it has been confirmed or not.	N/A, SQL queries will be run.	N/A, SQL queries will be run.
Wallets	Stores wallet addresses.	N/A, SQL queries will be run.	N/A, SQL queries will be run.
Settings	Stores settings set for the user.	N/A, SQL queries will be run.	N/A, SQL queries will be run.

Ethereum Smart Contract Module:

Method Name	Purpose	Inputs	Outputs
transfer_between_wallets	Transfers DANC tokens between two wallets.	2 wallet addresses	Boolean, true if created, false if not created
verify_wallet_address	Verify wallet address	A single wallet address	Boolean, true if valid address, false if not valid
create_token	Creates tokens (ran only once on token creation)	Parameter for number of tokens to create	Boolean, true if created, false if not created
burn_token	Destroys tokens (in case of a ETH cryptocurrency fork, or attack on DANC token)	Parameter for number of tokens to burn	Boolean, true if burned, false if not burned

Blockchain to Backend Communication Module:

Method Name	Purpose	Inputs	Outputs
add_tx	Adds a tx from the blockchain to the database.	A transaction as a parameter	Boolean, true if added, false if not added
send_tx	Sends an amount and an address, and the blockchain sends out DANC tokens for that amount to an address.	Amount and address	Boolean, true if sent, false if not sent
is_valid_tx	Checks to see if a transaction is valid.	Transaction as a parameter	Boolean, true if valid address, false if not valid
is_confirmed_tx	Checks to see if a transaction is confirmed	Transaction as a parameter	Boolean, true if confirmed, false if not confirmed

Frontend to Backend Communication Module:

Method Name	Purpose	Inputs	Outputs
get_user_balance	Gets the user's balance.	User ID	Returns the user's balance
get_top_10_tx	Gets the users top 10 transactions	User ID	Returns the top 10 highest transactions by value
get_recent_tx	Gets the users recent transactions	User ID	Returns the 10 recent transactions by time
login	Login	None	JWT, user ID
register	Register	None	JWT, user ID
forgot_password	Forgot Password	Email	None
handle_oauth	Handle OAuth	Oauth Provider	JWT, user ID
send_transaction	Send Transaction	2 wallet addresses	Boolean, true if created, false if not created
verify_wallet_address	Verify wallet address	A single wallet	Boolean, true if valid

		address	address, false if not valid
create_new_wallet_address	Create a new wallet address for a user	User ID	Boolean, true if address created, false if not created
update_settings	Updates user settings	User ID, list of settings updated	Boolean, true if all updated, false if any not updated
get_settings	Gets user settings	User ID	List of settings

Validation Module:

Method Name	**Purpose**	**Inputs**	**Outputs**
generic_validation	A generic validation function that returns a message and an HTTP code.	Message, HTTP error code, error	Boolean, HTTP Code for valid / invalid
handle_validation	Runs a list of validators passed in and verifies that all validations are valid.	Lambda function, data	Boolean, HTTP Code for valid / invalid
return_list_of_validators	Returns a list of validators used.	None	Returns list of validators

Blockchain API library Module:

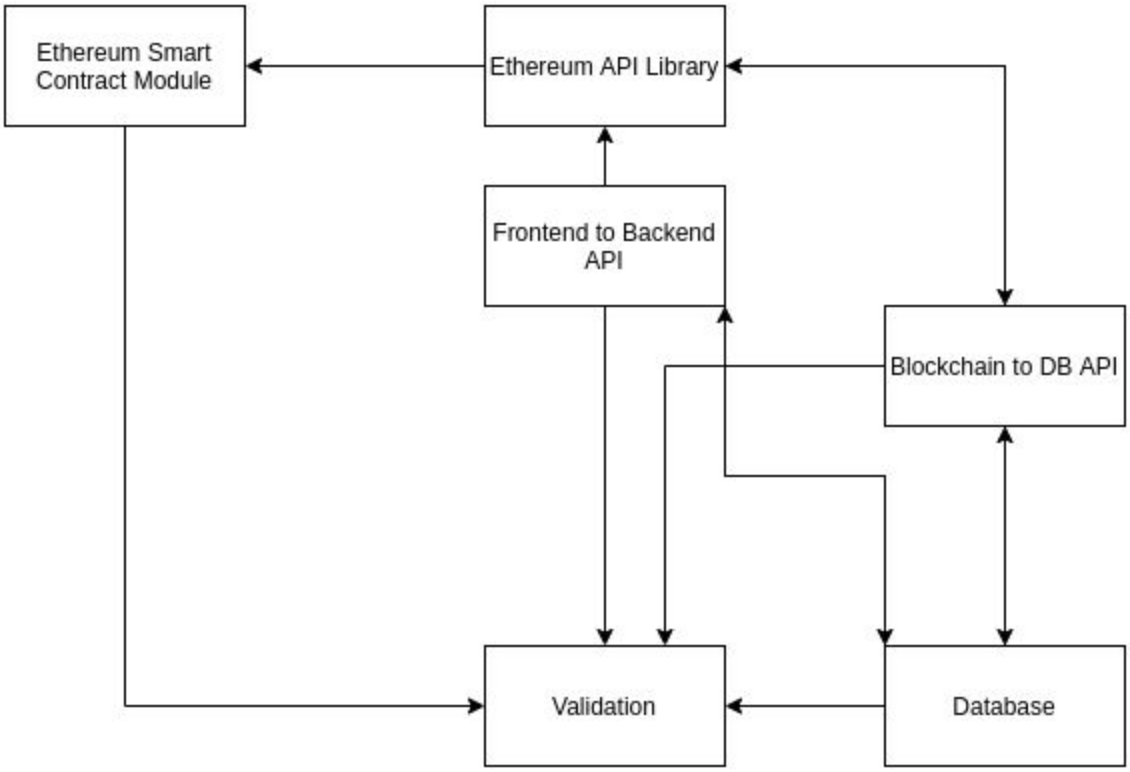
Method Name	**Purpose**	**Input**	**Output**
web3.eth.subscribe	Subscribe to events from DANC smart contract events	Type, smart contract address, event, callback function	Error, result
web3.eth.Contract	Interact with DANC smart contract	Smart contract address	None
[contract name].[smart contract method name].call	Calls a method of the DANC smart contract	Smart contract method parameters, gas amount, callback function	None

[contract name].[method name].estimateGas	Gives estimate GAS of method execution	Smart contract method parameters	Gas amount
web3.eth.accounts.create	Generate eth account	Random string	object: Address, privateKey, signTrasnaction, sign, encrypt
web3.eth.accounts.privateKeyToAccount	Generate eth account from a private key	privateKey	object: Address, privateKey, signTrasnaction, sign, encrypt

6. SRS Changes

1) Product Perspective - added functionality to backend
2) Product Functions - added user feature, view history
3) Apportioning of Requirements - added further user database information
4) External Interfaces - added Data needed and data returned
5) Functional - Separated login and registration features
6) Performance - Removed contingency plan as we have not done sufficient research on the best way of handling the issue.
7) Added Validation as a software interface.

7. Diagram showing communication between modules



8. Algorithms

Top 10 transactions:

We could store the top 10 transactions for each user in a cache on the server, to return to the front end tean, to show to the users, instead to requerying the data on every page load, therefore decreasing database reads. When the webpage is refreshed, the database should not be queried for the user to pull the top 10 transactions again. Eventually, when a user has thousands or tens of thousands of transactions, it would cause significant strain to our server, especially if we have many concurrent users. To ensure the cache is valid, on every transaction for a particular user we maintain a minimum heap and use the heap to verify whether a transaction amount is greater than the top of the heap. If so, we pop the minimum and add it to the heap. This ensures much fewer database reads to our backend server.

Bundling transactions:

When sending out transactions to the ethereum network, instead of sending out each individual transaction, we can bundle transactions together choose the output address with its values. This would save us considerable amount on fees, while not impacting user time significantly. This will be done only if we receive multiple transactions per second, as otherwise users will need to wait significant amounts of time otherwise.

Unauthorized access detection:

When a user logs into / tries to send a transaction with an IP that is different, they will have to verify their identity by clicking a link in their email / entering a 2FA code. This would help deter unauthorized access to people's accounts.