# ////Etch PreFlight Test

October 2018

## Purpose

The demo is designed to show the following -

1. Smart contracts can be used to store bonus conditions.
2. The bonus conditions can be linked to a wallet.
3. The bonus conditions can be tested and a wallet paid.

## Implementation

A UI was designed to show the above with the following assumptions -

1. Metamask for wallet management.
2. Ropsten Test Network (Ethereum).
3. Remix - Ethereum IDE - to test solidity smart contract.
4. Digital Ocean servers.
5. MongoDB to store local login details (usernames and passwords).
6. Sendgrid to send emails to the users (alerts).
7. Node.js for the application logic (non-blockchain).
8. Javascript/HTML/CSS for the main UI controls and logic.

The Screens enable the following -

1. A bonus or reward can be added to the blockchain. This has a name which should be unique. The bonus has variable properties such as Type, Target, Amount Payable, Deadline, and Token. These are all stored on the blockchain in a smart contract.
2. A wallet is added manually and linked to an email address using the UI.
3. The wallets are then linked to the bonus via a screen in the UI. This enables a wallet to be rewarded.
4. A myBonus screen allows a user to see their own bonuses and when they are due.
5. A general screen allows a bonus condition to be checked to see if the terms were met.

## Testing

The screens were tested using Remix to confirm the data, MongoDB (direct interface) to check login details, and the Ropsten network with Etherscan to test the blockchain transactions. Also, Metamask was used to confirm the transactions.

# Future Enhancements

## Purpose

To take a demo which showed the blockchain could be used and to address some issues in a real app -

1. Security
2. Scalability
3. Privacy
4. Robustness
5. Usability

## Security

Some basic level of security was implemented using session variables to store login details. The level of security can be enhanced as follows -

1. Https to create TLS txns.
2. Function modifiers in solidity to control who can access the functions.
3. ML to identify wayward txns (unusual txns), eg a large bonus.

The present design is based around wallets. Security can be improved by implementing interfaces to well-known wallets such as Tezos, Nano and Metamask - in a similar way as done by Myetherwallet. This would allow a more transparent journey for the user.

Also in this context ////Etch could create a wallet which would allow management of all bonuses and payments.

The question of which blockchain to use is not decided. Ethereum is used currently. If that changed to a cross chain solution, or using another chain such as EOS then specific security could apply, especially with regard to connection to a specific node on the blockchain (eg a local node).

As is commonly used by banks, eg a PIN reader, and many payment systems, a hardware device can be used to confirm identity. This adds an extra layer of security. Etch should use such a device to confirm key roles (such as adding a bonus).

## Scalability

Ethereum is known not to scale well. A full dapp would consider this question as follows -

1. Polkadot
2. EOS
3. Other emerging options (eg Tendermint)

These solutions are PoC and hence a degree of testing is needed to confirm what they can provide.

EOS has stated that scaling is possible to millions of TPS. Hence that could be an option. Polkadot offers a solution allowing cross chains and hence Etch could seamlessly integrate into other bonus/payroll related chains which could allow ////Etch to develop more generously.

## Privacy

This is an important issue. To achieve more privacy users may choose certain coins for bonus payments and also to be identified only by their wallet addresses.

////Etch should, therefore, store minimal data and work mainly via wallets with ////Etch login screens being a mask for wallets.

To protect privacy we may store data in a private database, and then store only hashed data onto the blockchain.

## Robustness

Due to the pace of change in the market, any solution should use reliable technologies. There are certain chains which will probably survive and hence ////Etch should focus on them.

# Usability

////Etch is by design an interface to a blockchain based solution. Hence the blockchain smart contracts need to designed so that they can survive changes. In Ethereum that will mean interfaced based contracts with an underlying contract to hold storage.

For EOS, the solution would be C++ based.

Also for a larger system, the question of storage should be examined. This could involve using a file system such as IPFS and also building a distributed search server such as Elasticsearch.

The underlying issue, is that the core data is all wallet based and hence file stores and search servers need to be layers on top of a more fundamental wallet layer. In other words, any data stored in JSON in a file system outside the blockchain should be hashed and then referred back to the blockchain.

For large txn processing, then batch processing would be implemented to check wallets against bonus conditions. That would involve for example a daily job running on ////Etch servers.

To check the blockchain contracts against existing reported data from users (eg via APIs or secure access to user databases). This design would be implemented in conjunction with the cross chain designs already referred to in this document, and a user blockchain could be referenced directly (eg user time logging system) by ////Etch batch processing to update Etch contracts with what bonuses should be paid and when.

A further enhancement would be for ////Etch to directly make the decision about paying a bonus, and then actually to pay the bonus. That would require ////Etch to have the authority to debit a user account to pay a bonus.