

Milestone-1: Course Project

CS - 2361 -- Blockchain and Cryptocurrencies

Prof. Mahavir Jhawar

Nandini Agrawal, Prashanthi R, Shivam Agarwal

Project Description

Universities and organisations across the globe promote top quality research by reserving a large amount of funds received from government, corporations and philanthropists for the purpose.

The aim of this project, *Hyperfunds*, is to ensure that the faculty at Ashoka University spend their *fixed* reserve of funds reliably with approval from the right parties - usually administrators or supervisors - using hyperledger.

In this project, we consider three types of parties -- the faculty, the Accounts department, and the Dean of Research. An amount of funds is earmarked for each faculty member every academic year by the Dean of Research. Funds that remain unspent by the end of an academic year *do not carry on* to the next. If a faculty member has Rs. 1,00,000 in the beginning, every time they want to spend an amount lesser than a **fixed threshold**, say, $x \leq 1,00,000$ (let $x = 40,000$ here), they need to get it approved by the Accounts department. However, every time they want to spend an amount that exceeds the 40,000 threshold, they would require an approval from the Accounts department as well as the Dean of Research. It is important to note that neither the Accounts department nor the Dean of Research is allowed to spend any of the funds earmarked for faculty. Thus, we also note that, in every spend-request either two of parties

(faculty and Accounts department) or all three of the parties are involved (faculty, Accounts department, and the Dean of Research).

Project Goals

The goal of this project is to ensure that the funds are always spent by the right parties with approvals from the required parties as described in the project description above. We plan to use Hyperledger fabric to achieve the same. Hyperledger also provides immutable storage, that allows us to have a trustworthy record of all the transactions involving the funds.

Why Blockchain for this project

The Blockchain network establishes trust, accountability and transparency, which are key to our project goal. We want the spending to be as transparent as possible to make sure nobody violates the rules of this system. Having an immutable storage allows accountability of the expenditure and maintains trust between the parties.

Project Flow and Technical Details

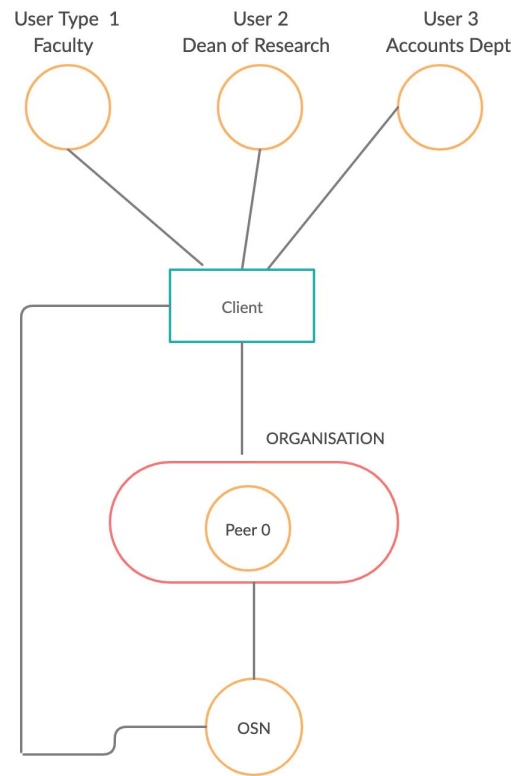


Figure 1: Network Structure for Hyperfunds

For the purpose of this project, as the Figure 1 above depicts, we plan to work with a network with one client, one organisation (with one peer) and one OSN. The three types of users -- faculty, Accounts department, and the Dean of Research -- are connected to a central client that sends out all transactions to the network. For now, we are considering only one Accounts department and one Dean of Research at Ashoka. However, the number of faculty users can be as many as we wish to have.

Each user has a dashboard that has functions based on what type of user they are. All users can sign in using their Ashoka email addresses. The Dean of Research and the Accounts department

can only register once with predefined email addresses - {dean.research@ashoka.edu.in,
accounts@ashoka.edu.in} to avoid any malicious activity.

Each faculty member too can register only once, but their email addresses are not predefined. On registering as a faculty member, a wallet is created. Funds are initiated by the Dean of Research. The faculty dashboard will have a provision to request the spending of any amount from their balance funds for research purposes. On sending the request, a transaction request will be created and sent to the rest of the network through the client. This transaction is called the ‘**proposal transaction**’. Additionally, at a later point, we hope to include in the transaction the hash of any document relevant to the proposal -- payment advice, bills, or other proofs of expenditure.

On the other hand, the Accounts department has a dashboard that will have options to query all the proposals made until that point in time and query transactions made by faculty members by proposal transaction ID. This dashboard will also have an option to approve the proposals. If this user approves any of the proposals, another transaction is created. This will be called an ‘**approval transaction**’. The approval transactions are also sent to the rest of the network through the client.

The dashboard of the Dean of Research will be exactly the same as the Accounts department. However, they will be able to approve only the requests that exceed the threshold. They also initiate funds to any faculty member’s account on registration. They can also use this

functionality to add funds to faculty members' accounts. Transactions that initiate / add funds will also require approval by the Accounts department.

Note: For a proposal, only when the right number of approvals is obtained from the right parties according to the rules (described in the Project Description section), will there be a change in the asset in the chainstate for that transaction.

Client Side Application Features

There will be multiple javascript files that will enable various features for the client-side application. We have described them below:

EnrollAdmin.js

This javascript file enrolls an admin to *Hyperfunds*. Needless to say, an admin is really important to manage any application like this.

RegisterUser.js

This javascript file registers users to *Hyperfunds*. All users must specify the type of user they are: faculty / Accounts department / Dean of Research. In our project, the Dean of Research and the Accounts department can *only register once* with pre-specified email IDs -- {dean.research@ashoka.edu.in, accounts@ashoka.edu.in}. However, any number of faculty members can register with different email IDs.

Note: The Dean initiates funds to the faculty members' wallet. This balance is updated on the client side after every successful transaction. It goes without saying that the Accounts department and the Dean will be able to view the updated balance of every faculty.

Balance.js

This javascript file provides the functionality that deals with the balance of faculty members. It is described below:

Get Balance -

```
( 'getBalance' , Email ID)
```

The client can use this to view the current balance of faculty members on all the three dashboards. This functionality will invoke the function `getBalance()` in the chaincode, which will return the balance of the faculty whose email ID is passed as argument.

Note: faculty members will have access to only their current balance, unlike the other two types of users.

Query.js

This javascript file provides the features to - **query transactions based on transaction IDs** and **query all transactions that have been made**. These have been described below:

1. Query -

```
( 'QueryTransaction' , Transaction ID)
```

The client passes the **transaction ID** of the transaction that they want to query.

The chaincode has a function **query()** that will return the right transaction. It is important to note that a faculty member will not be able to query a transaction that has not been made by them. If a faculty member tries to do this, an error should be thrown. However, the Dean of Research and the Accounts department can query and view any transaction by their transaction ID.

2. Query All -

('QueryAllTransactions')

Using this functionality, the client will be **able to view all transactions that have been made in the network so far**. Here, it is important to note that using this functionality, a faculty member will only be able to view all the transactions made by them. They will not be able to view all the transactions made by other faculty members. However, the Dean of Research and the Accounts department can query all and view all transactions.

Invoke.js

This is the javascript file that provides two key features to our application - **creating a proposal transaction** request for the faculty and **creating an approval transaction** request for the other two types of users. The types of transactions have been described below --

1. Create Proposal Transaction -

`('CreateProposalTransaction', Proposed amount, Email ID)`

The client passes the proposed amount along with the faculty member's email ID.

This method will send a request from the client side. Every proposal is added to the chainstate as a key-value pair. The transaction will not be processed until it receives enough approvals.

The Dean of Research can create a transaction to **initiate or add funds** to any faculty member's account. **Proposed amount** is passed as a positive value in this case, since it has to be added to the existing balance. This transaction would require the **Accounts department's approval**.

This function is also used by faculty members to create a transaction and **spend their funds**. **Proposed amount** is passed as a negative value in this case, since it has to be subtracted from their existing balance. As also mentioned previously, this transaction would require approvals from the Accounts department and the Dean of Research based on conditions described in the Problem Description section.

2. Create Approval Transaction -

`('CreateApprovalTransaction', Transaction_ID)`

The second argument in this transaction is the transaction ID that has to be approved. This will send a request from the client side through the Dean / Accounts department's ID, and the chaincode will proceed to update the faculty member's balance only if the approvals meet the conditional requirements.

Asset for the application:

The structure of the asset of our chainstate is as follows:

Key	Value
{Transaction ID}	Transaction Object: {Faculty ID, Proposed amount, Approval Counter, List of Approvers, Email ID}

The value part of the asset is a tuple with:

Faculty ID: This will be a unique ID assigned to each faculty member in the chaincode.

Proposed amount: The amount the faculty member wants to spend from their balance / The amount the Dean wants to add to the faculty member's account. It is a positive value when the Dean makes a transaction and a negative value when the faculty members make a transaction.

Approval Counter: The number of approvals that the transaction has got. It should be 1 or 2 based on conditions described in the problem statement. This will be set to -1 when the required number of approvals are reached, in order to prevent more approvals that might lead to confusion.

List of Approvers: This will basically be either [Accounts department] or [Accounts department, Dean of Research] based on the conditions in the chaincode.

Email of faculty: The email ID of the faculty member who is making the transaction.

Chaincode and Chainstate

The chaincode is the most important aspect of the application. It will possess a list of rules that *Hyperfunds* will work with.

Here are the global variables that we will use in the chaincode:

1. The global dictionary **balance_dict** is created for our use so that every transaction made by the faculty member uses the updated current balance. This dictionary has **faculty_email_ID** as key and **Current_Balance** as value.
2. **txn_id** is also made global so that each transaction gets a unique key value.

There are multiple functions that the chaincode for this application will have. Below, we have defined these functions:

getBalance():

Multiple other methods update the global dictionary `balance_dict`. This method returns the balance of a faculty based on the given email ID. We should ensure that the **faculty members should not be able to access the balance of other faculty members**. A simple pseudocode for this method would be:

```
getBalance(email_ID):  
    return(balance_dict[email_ID])
```

`CreateProposalTransaction()`:

The list of arguments this method receives are: `{Proposed amount, Email ID}` of a faculty member. This method will create a proposal transaction with the given Email ID of the faculty member, the proposed amount. Here is the list of things this method will have to do:

1. Get the balance of the faculty using the `getbalance()` method and check if the proposed amount plus the balance obtained from the `getbalance()` is a positive value.
 - If no, throw the error 'Insufficient Funds! Transaction Denied.'
 - Exit the if condition.
2. Assign this transaction a `txn_id`.
3. Create an asset instance (key-value pair) with the `txn_id` as key and `{Faculty ID, Proposed amount, Approval Counter, List of Approvers, Email ID}` as value and push it to the chainstate.

`CreateApprovalTransaction()`:

The arguments that this method gets are: `{Transaction_ID}`. This method will invoke a transaction whenever the Accounts department or the Dean of Research approve a transaction on their dashboard. The changes that this method makes to the transaction with ID

`Transaction_ID` are given below:

1. Add the approver's ID to `approver_list` .
2. Increment the value of `approval_count` by 1. The default value of this variable is 0.
3. Check if the condition for the number of correct approvals is met. If yes
 - Check if the faculty has sufficient funds remaining in order to make this transaction.
 - If not then display 'Insufficient Funds! Approval Denied'
 - Set `approval_count` to a large negative value.
 - Exit out of the outer if condition.
 - Make `approval_count` = -1. This is done in order to mark that this transaction has been appropriately approved..
 - Update the global balance dictionary `balance_dict` with the value
`(Current_Balance + Proposed amount)`.
4. Push updates to the chainstate.

`QueryTransaction()`:

This function allows us to query based on the Transaction ID. A faculty member will not be able to query a transaction that has not been made by them. If a faculty member tries to do this, an error should be thrown. However, the Dean of Research and the Accounts department can query and view any transaction by their transaction ID.

QueryAllTransactions():

This function allows us to query all transactions. A faculty member will only be able to view all the transactions made by them. They will not be able to view all the transactions made by other faculty members. However, the Dean of Research and the Accounts department can query all and view all transactions.
