**DragonCoin API**

As a reference, here are all of the possible GET and POST requests in our API

For implementation for class credit, you will probably only be concerned with the transaction method, though.

Code located at: <https://github.com/Siryu/blockServer>

To get started, take a look at the random transaction generator. It will generate random transactions and send them to the system; for better interfacing, we would expect a client to actually allow users to create transactions at will.

1. To start your own pool
   1. Install node.js on your system if it’s not already there.
   2. Run poolLeader.js with paramaters to connect as follows
      1. Node poolLeader.js –localPort [5 digit number]
   3. This will allow you to create your own instance of a Pool to test with
   4. (not required) To connect another pool to this pool
      1. Node poolLeader.js –localPort [5 digit] –remoteIp [http://ip of other pool] –remotePort [5 digit of other pool]
2. There is a version of the server available also here:
   1. DragonCoin.herokuapp.com
   2. Use it to avoid connection issues with the Neumont network.

**Route:** {root}/api/…

**POST** /transaction

* Description: Tell the block chain that you would like to spend money  
  You send a block that you would like to spend, the address of the person you are sending the money to, the amount of money from the block to spend (up to the amount the block contains), and the time of the transaction. If the entire block is not sent, the remaining amount will be sent back to you in the form of a new block
* Requires: JSON

{

sendingBlockAddress: [blockToSpend],

receivingIPAddress: [location for the money to be sent],

amount: [amount],

time: [time of transaction]

}

* Returns: Status Code and a json object if accepted of the form:  
  {  
   blockNumber: [block number]  
   amount: [amount of money in the block]  
  }

**GET** /work

* Description: if there is no block header provided, will return the first block in the chain  
  if there is one provided and it is in the block chain, it will either return the next in the chain, or itself if it is the head of the chain  
  if the provided block is not in the chain, it will provide a 404 not found and a message stating the block was not found
* Requires: nothing
* Optional: a query string parameter with the block header of your most recent block
* Returns: JSON of the block

**POST** /solution

* Description: This will send a verified block with its solution to the server; if it is not actually verified, the server will send back a statement saying the block was not accepted
* Requires: JSON

{

blockWorked: [block],

solution: [solution],

nonce: [nonce]

}

* Returns: Status Code

**POST** /subscribe

* Description: will register the sent information with the server as an interested party to receive updates
* Requires: JSON

{

name: [name],

address: [address],

port: [port]

}

* Returns: Status Code

**POST** /unsubscribe

* Description: will unregister the host name so it will no longer receive updates
* Requires: JSON

{

name: [name]

}

* Returns: Status Code