Blockchain RESTful API Specs

Cybersecurity Protocol and Mechanism for e-Logistics of Dangerous Goods Tracking using Blockchain

Sumanta Bose,* Mayank Raikwar,§ Jiale Guo, Haoze Qiu
Sourav Sen Gupta, Anupam Chattopadhyay,† Kwok-Yan Lam‡

*sumanta001@e.ntu.edu.sg, §mraikwar@ntu.edu.sg, †anupam@ntu.edu.sg, ‡kwokyan.lam@ntu.edu.sg

School of Computer Science and Engineering, Nanyang Technological University, Singapore

Contents

The Blockchain RESTful API is software-as-a-service that allows an user (client) to invoke a host (server) running a blockchain enabled ledger – to maintain, retrieve and search supply chain transaction records, that are securely logged in into the blockchain. The API functions as a remote web service which allows users to use our blockchain enabled distributed ledger platform from a website, software or mobile application. The API gives access to all of the supported functions of our service. The web service uses 'REST-like' RPC-style operations over HTTP POST requests with parameterized URL encoded into the request and its response is encoded in JSON. It is designed to be easy to use and users can implement it in any model computer language that allows them to generate TCP requests.

1 Introduction

Blockchain enabled supply chain transaction management can help participants to effectively record price, date, location, quality, certification, and other relevant information. The availability of this information in the blockchain can increase traceability of material supply, lower losses from counterfeit and gray market, improve visibility and compliance over outsourced contract manufacturing, and potentially enhance an organization's position in responsible manufacturing. Blockchain driven innovations in the supply chain will have the potential to deliver tremendous business value by increasing transparency, reducing risk, and improving overall supply chain efficiency.

The Blockchain RESTful API is software-as-a-service, which facilitates users to build and implement a supply chain management system with the aforementioned benefits. Users can invoke a host running a blockchain enabled ledger to post, view and query transaction records, which are secured in the blockchain.

2 Functionalities

Currently, the supported API functionalities are: setting up the blockchain platform, posting transactions, viewing transaction list and entire blockchain, and querying transactions based on some transaction field and value.

In the Blockchain platform, transactions can be posted by the user, which are saved in the database. Based on some user-defined criteria, the system periodically collates a number of transactions and logs them into sequential blocks, forming the blockchain.

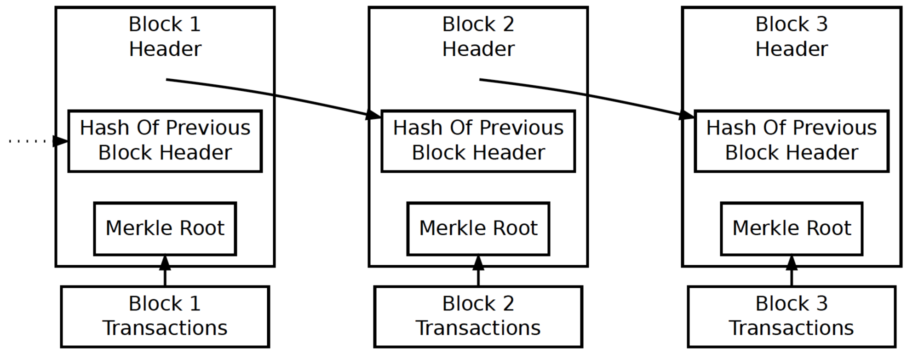

Figure 1: *A Simplified Blockchain structure.*

3 Definitions

Here we formally define transaction and block, and their JSON components of their data structure.

Transaction

A transaction is the agreement between two or more parties over the transfer of certain goods or currency. There can be other additional attributes to a transactions such as `order id`, `company id`, etc. For our Blockchain platform, Table 1 gives the JSON components of the transaction data available through the RESTful API.

JSON component	Description
`code`	Transaction Code
`shipCompId`	Shipping Company ID
`receiptCompId`	Recipient Company ID
`status`	Transaction Status
`shipDate`	Shipping Time
`receiptDate`	Received Time
`productDetails`	Product Description
`productId`	Product ID
`num`	Dispatched Quantity
`receiveNum`	Received Quantity
`OrganizationId`	ID of Organization who created the transaction

Table 1: *JSON components and description in Transaction data.*

The data structure of *Transaction* is as follows.

```
type Transaction struct {
      SerialNo int ‘json:"serialNo"‘
      Timestamp string ‘json:"timestamp"‘
      Code string ‘json:"order_id"‘
      ShipCompId string ‘json:"shipCompId"‘
      ReceiptCompId string ‘json:"receiptCompId"‘
      Status int ‘json:"status"‘
      ShipDate time.Time ‘json:"shipDate"‘
      ReceiptDate time.Time ‘json:"receiptDate"‘
      ProductId string ‘json:"productId"‘
      Num int ‘json:"num"‘
      ReceiveNum int ‘json:"receiveNum"‘
}
```

Block

Transaction data is permanently recorded in objects called blocks. Thus a block is like a ledger or record book. Blocks are organized into a linear sequence over time (also known as the blockchain). For our Blockchain platform, Table 2 gives the JSON component of the block data available through the RESTful API.

JSON component	Description
`BlkSNo`	Block Number
`Transaction Array`	List of Transactions in the block
`No of Transactions`	No of Transactions in the block
`PrevBlkHash`	Hash of Previous Block
`BlkTimestamp`	Timestamp of the block
`AgentId`	Id of Agent who created the block

Table 2: *JSON components and description in Block data.*

The data structure of *Block* is as follows.

```
type Block struct {
      BlkSerialNo int ‘json:"blk_serialNo"‘
      Timestamp string ‘json:"transaction_timestamp"‘
      MapOfTxHash map[int]string ‘json:"transaction_list"‘
      NoOfTx int ‘json:"transaction_count"‘
      PrevBlkHash string ‘json:"prevBlkHash"‘
}
```

4 Methods

The four available methods in our Blockchain RESTful API are as follows.

- `SETUP`. This is used to setup the hyperparameters of the blockchain system.
- `POST`. This is used to post new transactions.
- `VIEW`. This is used to view existing transactions and the blockchain.
- `QUERY`. This is used to query transactions for a particular field and value.

5 Service Access Point

The service access point URI is `http://155.69.202.34:8080`.

6 Service Endpoints

All Blockchain REST endpoints have the following format.

URI: `.../{method}/{target}/.../{list of parameters}`

The `{method}`, as discussed in Sec. 4 can be either of the following four. It determines the action of invoking the REST endpoint.

- `setup`
- `post`
- `view`
- `query`

The `{target}` can be either of the following two. It tells if the targeted object of action is transaction or blockchain.

- `transaction`
- `blockchain`

The `{list of parameters}` may or may not be present depending on the task at hand. This is described in detail in Sec. 7.

7 Endpoint Parameters

A REST endpoint provides way to map a URI and HTTP method simply and directly into the API framework. All the service endpoints URIs must be be suffixed after the service access URI ie. after `http://155.69.202.34:8080`. A descriptive list of the available service endpoints along with the endpoints URI are given below. In subsequent URIs, `...` refers to the service access URI. Readers may refer to Sec. 8 to see examples on how to use the service endpoints.

Landing

The landing endpoint is a static information page about the Blockchain RESTful API. The URI is as follows.

URI: `.../`

Setup

The setup endpoint allows users to setup the following two hyperparameters of the blockchain system.

- Block generation scheme. This hyperparameter is used to fix whether a *block* should be generated after a pre-decided *timeout* occurs or a pre-decided *number of transactions* are posted.
- Block generation trigger. Based on the block generation scheme, this hyperparameter is used to fix what the pre-decided *timeout* should be, or how many *number of transactions* (*size*) must be posted, for a block to be generated.

The URI to setup the blockchain system is as follows, where the parts in blue is fixed, while the parts in red are user input fields.

URI: `.../setup/blockchain/{scheme}/{trigger}`

Based on the user's requirements the values of `{scheme}` and `{trigger}` must be as follows.

- `{scheme}`. This should either be `timeout` (for pre-decided *timeout*) or `size` (for pre-decided *number of transactions*) for block creation.
- `{trigger}`. This should be a positive non-zero integer (n). If `{scheme}` = `timeout`, blocks will be generated after n seconds, and if `{scheme}` = `size`, blocks will be generated after n transactions have been posted.

The default setting is `{scheme}` = `timeout` and `{trigger}` = `5`.

Post

The Post endpoint allows users to post ie. save transactions in the ledger, which the system securely logs in into the blockchain. The URI to post transactions is as follows, where the parts in blue is fixed, while the parts in red are user input fields.

URI: `.../post/transaction/manual/{code}/{shipCompId}/{receiptCompId}/{status}/{shipDate}/{receiptDate}/{productId}/{num}/{receiveNum}`

Based on the user's requirements the values of the user input fields must comply to the following.

- `{code}`. Order ID (Code). Data type = *string*.
- `{shipCompId}`. Shipping Company ID. Data type = *string*.
- `{receiptCompId}`. Recipient Company ID. Data type = *string*.
- `{status}`. Transaction Status. Data type = *int*.
- `{shipDate}`. Shipping Date. Data format = DD-MM-YYYY.
- `{receiptDate}`. Receipt Date. Data format = DD-MM-YYYY.
- `{productId}`. Produce ID. Data type = *string*.
- `{num}`. Shipped quantity. Data type = *int*.
- `{receiveNum}`. Received Quantity. Data type = *int*.

Alternately, the user can post a auto-generated transaction, with randomly generated entries to the transactions fields. This can be used for testing or demo purposes. The URI to post an auto-generated transaction is as follows.

URI: `.../post/transaction/auto`

For posting several auto-generated transaction, with randomly generated entries to the transactions fields, we have another endpoint. The URI to post several auto-generated transaction is as follows, where the parts in blue is fixed, while the parts in red are user input fields.

URI: `.../post/transaction/autoloop/{loopTimes}`

Based on the user's requirements the values of `{loopTimes}` must be any positive integer. The system will post those many random transactions at an interval of 550 milliseconds between each.

View

The View endpoints allows users to view ie. retrieve (*i*) a list of all the transactions, and (*ii*) the entire blockchain structure. The URI to view transactions and/or blockchain is as follows, where the parts in blue is fixed, while the parts in red are user input fields.

URI: `.../view/{choice}`

Based on the user's requirements the values of `{choice}` must be either (*i*) `transaction`, or (*ii*) `blockchain`. In the first case, a list of all the transactions will be returned in the JSON format. And in the second case, the entire blockchain structure will be returned in the JSON format.

Query

The Query endpoint allows users to query ie. search the list of all transactions against a particular transaction field for a field value.

URI: `.../query/transaction/{field}/{value}`

Based on the user's requirements the values of `{code}` and `{value}` must be as follows.

- `{field}`. This is the transaction field on which the search query is being made. This could be any of these – `code`, `shipCompId`, `receiptCompId`, `shipDate`, `receiptDate`, `productId`, `num`, `receiveNum`.
- `{value}`. This is the value of the `{field}` on which the search is being conducted.

Clear

The Clear endpoint allows users to clear (delete) all existing transactions and blockchain records posted thus far. This can be used if and when necessary to clean up junk transactions.

URI: `.../clear`

Stop

The Stop endpoint allows users to stop the entire blockchain platform server. In the current release version, there is no feature of remotely starting the server again. Only server administrators can start the platform.

URI: `.../stop`

8 Examples

Table 3 shows some examples of service endpoints usage.

REST endpoint	Action
`.../`	To view the landing information page.
`.../setup/blockchain/timeout/10`	To setup the blockchain system with block generation timeout = 10 seconds.
`.../setup/blockchain/size/5`	To setup the blockchain system with block generation by number of transactions = 5.
`.../post/transaction/manual/AX7005/COMP1/COMP2 /5/01-02-2018/05-02-2018/QR2172/200/150`	To post a manual transaction with the transaction fields as given.
`.../post/transaction/auto/`	To post an auto-generated transaction with the transaction fields randomly generated.
`.../post/transaction/autoloop/5`	To post five auto-generated transaction with the transaction fields randomly generated, at an interval of 550 milliseconds.
`.../view/transaction/`	To view the entire list of transactions.
`.../view/blockchain/`	To view the entire blockchain.
`.../query/transaction/code/AX7005`	To query and view the list of transactions with code = AX7005.
`.../query/transaction/productID/QR2172`	To query and view the list of transactions with productID = QR2172.
`.../clear`	To clear (delete) all existing transactions and blockchain records.
`.../stop`	To stop the blockchain platform server.

Table 3: *Examples of service endpoints usage.*

9 Database Structure

The Transaction Table in the MySQL Database, storing the list of transactions has the following description.

```
+----------------+-------------+------+-----+---------+----------------+
| Field          | Type        | Null | Key | Default | Extra          |
+----------------+-------------+------+-----+---------+----------------+
| TxSNo          | int(11)     | NO   | PRI | NULL    | auto_increment |
| code           | varchar(50) | YES  |     | NULL    |                |
| shipCompId     | varchar(20) | YES  |     | NULL    |                |
| receiptCompId  | varchar(20) | YES  |     | NULL    |                |
| status         | int(11)     | YES  |     | NULL    |                |
| shipDate       | varchar(60) | YES  |     | NULL    |                |
| receiptDate    | varchar(60) | YES  |     | NULL    |                |
| productId      | varchar(20) | YES  |     | NULL    |                |
| num            | int(11)     | YES  |     | NULL    |                |
| receiveNum     | int(11)     | YES  |     | NULL    |                |
| OrganizationId | varchar(30) | YES  |     | iSprint |                |
| TxHash         | varchar(64) | NO   | UNI | NULL    |                |
| TxTimestamp    | varchar(30) | YES  |     | NULL    |                |
| BlkSNo         | int(11)     | YES  |     | NULL    |                |
| SeqNo          | int(11)     | YES  |     | NULL    |                |
+----------------+-------------+------+-----+---------+----------------+
```

The Blockchain Table in the MySQL Database, storing the parameters of every block of the blockchain has the following description.

```
+--------------+-------------+------+-----+---------+----------------+
| Field        | Type        | Null | Key | Default | Extra          |
+--------------+-------------+------+-----+---------+----------------+
| BlkSNo       | int(11)     | NO   | PRI | NULL    | auto_increment |
| StartTxSNo   | int(11)     | YES  |     | NULL    |                |
| EndTxSNo     | int(11)     | YES  |     | NULL    |                |
| PrevBlkHash  | varchar(64) | YES  |     | NULL    |                |
| ThisBlkHash  | varchar(64) | NO   | UNI | NULL    |                |
| BlkTimestamp | varchar(30) | YES  |     | NULL    |                |
| AgentId      | varchar(30) | YES  |     | Yap     |                |
+--------------+-------------+------+-----+---------+----------------+
```

10 Troubleshooting

The current version is a beta-release and has some usage guidelines and troubleshooting tips as follows.

- The `.../post/transaction/auto` URI requires some finite time to execute at the server side. Therefore, it is recommended to introduce some delay between subsequent calls. Around 500 to 550 milliseconds is safe.
- Based on this observation, we have introduced a new URI `.../post/transaction/autoloop/{loopTimes}`, which users can use to post multiple random transactions with one call. It is recommended to use this over repeatedly calling the former URI.
- The `.../view/{choice}` URI, is expensive on memory and time, since it retrieves the entire content of the database (Transaction Table and/or Blockchain Table) and passes it to the JSON container. Therefore, it is not recommended to call this a loop. This is suited for a one-time call (not in close succession), after posting a number of transactions. The time/memory expenditure increases with the size of the database tables.
- Based on this observation, we have introduced a new URI `.../clear`, which users can use to clear (delete) all existing transactions and blockchain records. It is recommended that this is called after posting about 100 transactions. This will ensure faster response time and speedy retrieval of JSON data.

These performance glitches will be improved and bugs fixed in subsequent stable releases.

11 Additional Notes

This Blockchain RESTful API was developed at Nanyang Technological University as part of the NRF project entitled '*Cybersecurity Protocol and Mechanism for e-Logistics of Dangerous Goods Tracking using Blockchain*' under project ID NRF2016NCR-NCR002-025 in collaboration with iSprint.