REST API

All REST API requests MUST be forwarded to the domain **http://kaboom.rksv.net** under the context **/api** hence http://kaboom.rksv.net/api/

It is advised to use **Accept-Encoding: gzip** HTTP Header to reduce take advantage of gzip compressed data.

CORS has been enabled at the Server and if you are facing CORS related issues, you are advised to use the CORS related standard HTTP headers.

Debugging
If needed, you can debug the API by visiting **http://kaboom.rksv.net/** in your browser and see in action, how the communication is established.

Status

The Status REST API can be used to poll the API status. If the API Server is correctly configured and is running then this API will respond with a 200 OK and a JSON Object with the key **status**, whose value MUST always have the value **OK**.

Details

Method	: HTTP GET
Context	: /
Query String	: none

Example

```
GET /api HTTP/1.1
Host: kaboom.rksv.net

STATUS: 200 OK

{
   "status": "OK"
}
```

Historical

The Historical REST API can be used to pull in historical records. The HTTP request can contain an optional query string **interval** whose value can range from 1 to 9. The **interval** will indicate the numbers of records to fetch, as described below

Interval	Number of Records fetched
1	200
2	400
3	600
4	800
5	1000
6	1200
7	1400
8	1600
9	1800

If the query string **interval** is not present or has value other than the one described above, then the entire dataset will be fetched, which contains about 2500 records.

The API will respond with a 200 OK and a JSON Array containing Strings of Comma Separated Values (CSV).

The CSV must be parsed as per the following header information -

timestamp,open,high,low,close,volume,

The timestamp is a Unix Epoch Timestamp.
The open, high, low, and close denotes the OHLC prices of that particular stock. Prices must always be considered as floating point values.
The **volume** denotes the volume of trade conducted during that period of time.

Details

Method	: HTTP GET
Context	: /api/historical
Query String	: interval=<INTERVAL>

Example

```
GET /api/historical?interval=1 HTTP/1.1
Host: kaboom.rksv.net

STATUS 200 OK

[
   "1308594600000,839.9,855,837.3,848.65,3980489,",
   "1308508200000,875,875,828.1,833.25,6926078,",
   "1308249000000,891.3,893.3,865.2,868.75,5035618,",
   "1308162600000,901,901,884.1,887.55,5707528,",
   "1308076200000,912,918.9,898.6,900.55,4607486,",
.
.
.
.
]
```

WebSocket API

The WebSocket API complies with Socket.IO standardizations and hence the client must also follwo the same. You are allowed to use any library that complies with Socket.IO communication standards.

All WebSocket API requests MUST be forwarded to the domain **http://kaboom.rksv.net**

The WebSocket API is grouped into Socket.IO **namespaces**.
@see for more details -
http://socket.io/docs/#restricting-yourself-to-a-namespace
http://socket.io/docs/rooms-and-namespaces/

All responses to the client are sent under two events - **data** and **error**.

Example:

```
clientSocket.on('data', function(data) {
	console.log('Response: ' + data);
});

clientSocket.on('error', function(error) {
	console.error('Error: ' + error);
});
```

/watch

Namespace : /watch
Events that Server listens to : sub, unsub, ping
Events that Client listens to : data, error

Event : ping

When the client emits the event ‘ping’, the server will respond with event ‘data’ containing the ‘pong’ data.
Payload:
{}

Event : sub

This event is used for subscribing to live stock quotes. The format of the data received is CSV with headers same as described in the Historical REST API section.

Payload:
{ state: true }

NOTE:
The data is pushed by the server every 100 milliseconds, but the data will only be sent to the client, if the client acknowledges the previous data. In other words, to continue receiving the data, the client must send acknowledgement.

The acknowledgement code is 1 (Integer).
var CLIENT_ACKNOWLEDGEMENT = 1;

@see for more details -
http://socket.io/docs/#sending-and-getting-data-(acknowledgements)

Event : unsub

This event is used for un-subscribing to live stock quotes. Once you unsubscribe, you will no longer receive the live data from the server.

Payload:

```
{ state: false }
```

Examples:

```
// Ping the server, the payload will be ignored, hence sending an empty Json object
clientSocket.emit('ping', {});

// While subscribing, the state must be true
clientSocket.emit('sub', {state: true});

// While unsubscribing, the state must be false
clientSocket.emit('unsub', {state: false});
```