TERBINE API OVERVIEW

Posted: June 17, 2016
Version 0.9 (draft)

Table of Contents

Contents

Overview

This document is a high level overview of the TERBINE Application Programming Interface (API). This document is not intended to be an exhaustive list of available interfaces or services, but rather a smooth introduction to any client who intends to use the API to interface with the TERBINE marketplace services.

API Categories

The TERBINE API can be broken down into the following categories.

- Organization / User Administration
- Organization / User Profile and Preferences
- Metadata / Schema Administration
- Rule / Filter Maintenance
- Ingestion Services
- Data Browsing, Search and Discovery
- Buying (Cart) and Payment Interface
- Reporting and Analytics
- Extraction and Delivery

Security

API security will be implemented using a combination of a security strategy Java Web Token (JWT) for authentication. When a user successfully logs in they receive a TOKEN back from the server with a limited lifetime. This token is exclusive to that user, expires after a period of time and must be passed in to all subsequent calls.

Role based security based on pre-defined TERBINE based roles will provide authorization to functionality and resources.

In addition wire based security will be implemented using SSL/TLS.

As needed, disk based encryption will provide “data at rest” based security.
All Personally Identified Information (PII) will be stored in an encrypted format and salted as needed.

TERBINE also will support machine to machine (M2M) interaction and identify for this will be accomplished with a combination of provisioned API Key Identifier and Public Key.

The majority of the API will be secured excluding basic browsing, login and system overview services.

SERVICE MODEL

The TERBINE API will be implemented using a REST model built on HTTP(s). Within the REST model the system is divided into a group of resources (users, metadata, etc.) and each is uniquely identified. Generally, but not exclusively, resources use a Universally Unique Identifier (UUID) which is a 128 bit value. An example of a UUID is **`fb9967f1-eadf-465f-b11e-cbeda2c62bdf`**. These will be allocated and managed by the TERBINE backend.

HTTP VERBS

REST built on HTTP(s) uses standard HTTP verbs for managing resources.

- GET – read a resource by id or group of resources by name.
- PUT – update a resource
- POST – create a new resource
- DELETE – delete a resource
- PATCH – this will have limited use and is used when a portion of a resource will be updated
-

EXAMPLES

GET

- *GET* `http://www.terbine.com/user/fb9967f1-eadf-465f-b11e-cbeda2c62bdf`
- *GET* `http://www.terbine.com/user/fb9967f1-eadf-465f-b11e-cbeda2c62bdf/securityanswers`
- *GET* `http://www.terbine.com/metadata/sample`

POST

- *POST* `http://www.terbine.com/user`
- *POST* `http://www.terbine.com/user/fb9967f1-eadf-465f-b11e-cbeda2c62bdf/securityanswers`

PUT

- *PUT* `http://www.terbine.com/user/fb9967f1-eadf-465f-b11e-cbeda2c62bdf`

- *PUT* `http://www.terbine.com/user/fb9967f1-eadf-465f-b11e-cbeda2c62bdf /metadata/eb9967f2-eadf-465f-b11e-cbeda2c62bdf`

DELETE

- *DELETE* `http://www.terbine.com/user/fb9967f1-eadf-465f-b11e-cbeda2c62bdf`
- *DELETE* `http://www.terbine.com/user/fb9967f1-eadf-465f-b11e-cbeda2c62bdf /metadata/eb9967f2-eadf-465f-b11e-cbeda2c62bdf`

RETURN AND ERROR CODES

HTTP Verb	CRUD Operation	Entire Collection (e.g. /users)	Specific Item (e.g. /users/{id})
GET	read	200 (OK), list of users. Use pagination, sorting and filtering to navigate big lists.	200 (OK), single user, 404 (Not Found), if ID not found or invalid.
POST	create	201 (Created) Request Body with mandatory information for all resources. response with new resources, ids and associated data.	201 (Created) Request Body with mandatory resource information. response with new resource, id and associated data.
PUT	update	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid for all resources in body	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid for specific resource
DELETE	delete	404 (Not Found), unless implemented delete the whole collection—not normally desirable.	200 (OK). 404 (Not Found), if ID not found or invalid for resource

As a general rule success of any operation can be checked by looking for a return code in the range of 200 – 299.

Any return code in the 300 – 399 range will denote a moved resource and will be used sparingly and be documented thoroughly at the specific API level.

Any return code in the 400 range denotes a client error. Most common examples of these are:

- 400 – Bad request. The correct method was used, but there was missing information in the body or malformed entity being passed.
- 401 – Unauthorized. User needs to be authenticated and use a correct API Key / Session token.
- 403 – User is Not Authorized. User not allowed to perform this operation.
- 404 – Not Found. Resource with this ID not found.

- 405 – Method Not Allowed. A resource was called with a form not allowed (e.g. correct form for a GET but not a POST).

Server errors will be denoted by HTTP status codes in the 500 range. Most common are:

- 500 – Internal Server Error – unexpected error at server level
- 503 – Service Unavailable – API or service not available.

Optionally, specific information may be passed back detailing what is missing or why the error occurred.

Header Information

All REST calls will require multiple header values be set.

Within TERBINE we have multiple common header values that may or may not be required dependent on the value. The majority of the services are protected by requiring an **Authorization** and an **X-API-KEY**.

There is also a possibility to set a header value from client as **X-TRACKING-ID.** This id will be used for all logging on the server for that request. The use of this would be if there is any client logging, the client would create a new id for each request made to the server and in this way logging entries from client through to server and back in response can be linked by this id. The tracking id is very useful for user error reporting and troubleshooting.

If the client does not pass a tracking id, one will be created on the server and passed back in the response header.

Data Exchange Format

All data is interchanged in the API via JSON. JSON is a simple text-based message format that is often used with RESTful Web services.

JSON Example

Below is an example with multiple markers in red added to the JSON. All characters in red are not part of the original JSON.

```
{
 "id": "98931ef4-c423-42f5-869c-88eae5b39dfa",
```

```
"orgId": "6d94ff88-4b41-41d0-b3a4-d14d89da0939",
"identifier": {
  "extId": "A634321"
},
"meta": {
  "name": "Medical Test Metadata",
  "description": "Medical Test Metadata Description",
  "version": "v1"
},
"dataset": {
  "extId": "SRC_EXT_ID1",
  "type": 30,
  "sensorInfo": {
    "id": null,
    "type": 44,
    "make": "make 123",
    "model": "model 123",
    "comment": "Specific sensor comment"
  },
  "schemaInfo": {
    "format": 6,
    "type": 81,
    "property": ";",
    "body": null
  },
  "comment": "Comment on sample dataset information"
},
"containers": [
  {
    "type": 130,
    "parentId": null,
    "extId": "CONT_EXT_ID",
    "legalType": null,
    "latitude": "40.078521",
    "longitude": "-74.078974",
    "altitude": "N/A",
    "address": "zip:08742",
    "startDate": "2015-12-03T04:17:36Z",
    "endDate": null,
    "name": "Container name",
    "description": "Container long description"
  }
],
"deliveries": [
  {
    "type": 22,
    "size": 32032223232,
    "hash": null,
    "deliveryDate": "2015-12-03T04:17:36Z"
  }
],
"owners": [
  {
    "startDate": "2015-12-03T04:17:36Z",
    "endDate": null,
    "id": "67b181ee-eff0-4066-9629-badb071e9aff",
    "type": 60,
    "comment": "Ownership comment here"
  }
```

```
  ],
  "categories": [
    {
      "name": "Utilities",
      "id": 3
    },
    {
      "name": "Information",
      "id": 9
    }
  ],
  "tags": [
    {
      "name": "Voltage",
      "id": 0
    },
    {
      "name": "Sensor",
      "id": 0
    }
  ],
  "legal": [
    {
      "type": 102,
      "comment": "Legal Information for COPYRIGHT",
      "startDate": "2015-12-03T04:17:36Z",
      "endDate": null
    },
    {
      "type": 101,
      "comment": "Legal Information for GOVERNMENT",
      "startDate": "2015-12-03T04:17:36Z",
      "endDate": null
    }
  ],
  "regulatory": [
    {
      "type": 141,
      "comment": "Regulatory Information for SAFEHARBOR",
      "startDate": "2015-12-03T04:17:36Z",
      "endDate": null
    }
  ],
  "relations": [
    {
      "type": 112,
      "referenceId": "574c1a6a-34a6-4575-94cb-26f0c4c376bb",
      "comment": "Relationship Sample",
      "startDate": null,
      "endDate": null
    }
  ],
  "events": [
    {
      "eventDate": "2015-12-03T04:17:36Z",
      "eventType": 90,
      "comment": "Event set for type INGESTED"
    },
    {
```

```
      "eventDate": "2015-12-03T04:17:36Z",
      "eventType": 91,
      "comment": "Event set for type PERSIST"
    }
  ],
  "gradings": [
    {
      "type": 120,
      "grade": "Gold",
      "comment": "Sample grading information"
    }
  ]
}
```

EXPLANATION OF COMMENTS

1. All JSON will be wrapped with an identifier of what the root type of the data consists of. In this example, we know with the wrapped value it is of type **metadata**.
2. UUID – an example of a UUID value, which will be widely used within TERBINE.
3. Standard date format – this is an example of a standard date format, ISO 8601 format and transmitted in UTC. It is up to client to display these in any localized format. Date only format will be YYYY-MM-DD.
4. Embedded data – this is sub entity of the owning item **metadata** called **schema**. It has a child relationship to the parent. Dependent on context it may also live as a top level resource.
5. Null Values – this is how non-existent values are transmitted. If passed in a PUT or POST they will overwrite existing value with the NULL value.
6. Embedded Quotes – this shows a string with embedded quotes and therefore the \ character is preceding them.
7. Array of Embedded Objects – When there is a 0 to N (or 1 to N) the embedded objects are placed in an array as shown here.
8. Array of Single Data Items – When there is a list of single items they are embedded in an array as shown here.

CLIENT DEPENDENCIES

TERBINE API is built completely client agnostic. All API interfaces have neither client specific information nor variations of the definitions dependent on client type.

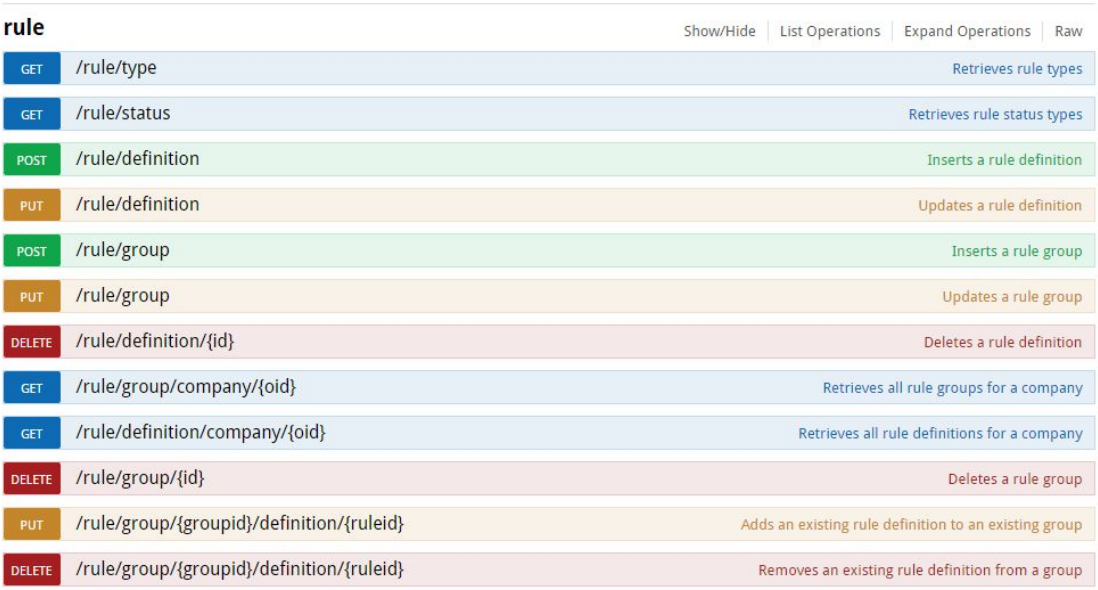
Therefore a Web client, iOS or Android client should find the use of the TERBINE API seamless.

Online API

All APIs will have an online blueprint which will allow clicking on a category of the API, getting a description of the API, parameters, possible return codes and even testing individual API services.

An example version to show the capabilities can be accessed at
http://52.33.108.29:8080/swagger

Another example is shown below for one specific category of services for RULES.



Data Model

Clients will not need to hard code or have embedded lists of items. An example is taken from above for rules. For instance, rules (or filters) can have one of three possible status in the system. The client does not need to hard code these, but rather calls the **`/rule/status`** end point and receives a list of IDs and descriptions for these items as shown in the *Response Body* section below.

Response Body


```
{
  "ruleStatus": [
    {
      "id": 1,
      "name": "VALID",
      "description": "Rule is Valid",
      "enabled": 1
    },
    {
      "id": 2,
      "name": "INVALID",
      "description": "Rule is Invalid",
      "enabled": 1
    },
    {
      "id": 3,
      "name": "DISABLED",
      "description": "Rule is Disabled",
      "enabled": 1
    }
```


Response Code

```
200
```

Response Headers


```
{
  "Date": "Mon, 07 Sep 2015 20:12:09 GMT",
  "Access-Control-Request-Method": "OPTIONS,GET,PUT,POST,DELETE,HEAD",
  "Vary": "Accept-Encoding",
  "Access-Control-Allow-Methods": "GET,PUT,POST,DELETE,HEAD",
  "Content-Type": "application/json",
  "Access-Control-Allow-Origin": "*",
  "Transfer-Encoding": "chunked",
  "X-Tracking-Id": "d696a39b-06e4-4602-b307-76d52f4c7064",
  "Content-Encoding": "gzip",
  "Access-Control-Allow-Headers": "X-Requested-With,Content-Type,Accept,Origin,Authorization,X-TRACKING-ID,X-SESSION-TOKEN"
}
```

REVISIONS

Version	Date	Name	Description
Initiated	2015/09/06	Brian Enochson	For Review
0.9	2016/01/20	Brian Enochson	Initial Internal Release