Contents

1 Purpose

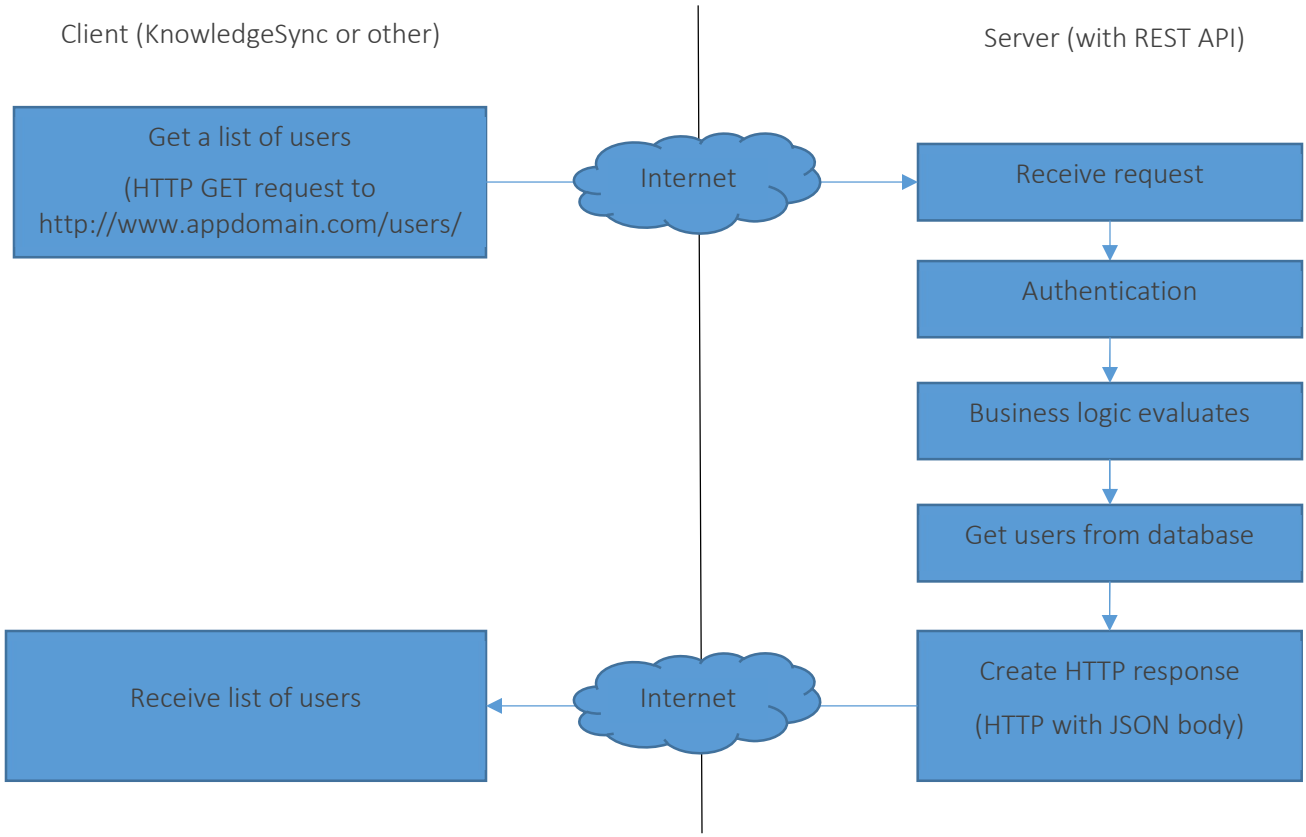
This REST API introduction provides a prerequisite basic introduction to the terminology and concepts needed for working with REST APIs. It is not a full reference on the subject. Please refer to the REST API triggers and REST API actions sections within the KnowledgeSync User Manual for specific information about using KnowledgeSync to interact with your web apps.

2 REST API Video

The REST API Concepts and Examples is an excellent overview of REST API.

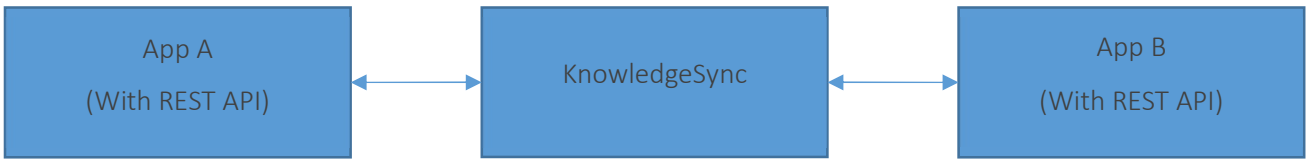
3 REST API Call and Response

The sequence diagram below shows a simple REST API call and response to get you familiar with the basic steps that occur to complete the call and response.



4 Connecting REST APIs

The diagram below shows how you can connect REST APIs with middleware such as KnowledgeSync. In this case KnowledgeSync might get data from App A and then transmit that data to App B. The communication between KS and App A and then the communication between KS and App B would follow the pattern shown in Section 6.3.



As an example, you might have KS get a list of users from App A and then submit the list of users to App B. In addition, the connections could be REST API or direct-database connections, or a mixture of the two.

5 REST API Benefits

REST APIs have grown tremendously in popularity because they enable consistent, safe access to retrieve as well as update data within web applications. Once you can safely and easily *retrieve and update data*, you can safely and easily *integrate* web applications.

For example, if you create an integration between an ERP and a scheduler:

Type	Traditional	REST API
Author	The integration is often written by the scheduler author and is therefore	The ERP author writes and tests the REST API code. The ERP author, scheduler

	ultimately dependent on their knowledge of the ERP and the testing and validation of the integration.	author, end user, or a third party will configure middleware to build the integration.
Business logic for creating, updating, and deleting data	Lives in the scheduler code.	Lives in the ERP code.
Data integrity	Scheduler author might introduce possible corruption to the ERP database. If the scheduler must update the ERP, database write access must be provided to the scheduler. This leaves a pathway to potentially corrupt the ERP database.	The ERP vendor creates the business logic to interface with the application and can therefore guarantee data integrity.
Integration location	Integration executable files are collocated with the ERP system files.	The integration might run on the same server as the ERP, but often is run on an integration middleware server.
Authentication and permissions	Defined using a database login, which is often not possible.	Clearly defined in the API.
Reusability	Integrations are written for two software packages.	The REST API remains the same and the integration middleware is the flexible connector to any number of software packages.
Integration updates when the ERP code is updated	ERP author must notify scheduler author and coordinate updates. Scheduler author must update the integration. If updates are uncoordinated or end-user fails to correctly install the update, data corruption may occur.	ERP author must notify scheduler author and coordinate updates. ERP author updates the REST API. If updates are uncoordinated or end-user fails to correctly install the update, an error may occur but no data corruption.

6 JSON[1]

To understand how REST APIs work, it helps to first understand JavaScript Object Notation (JSON). JSON is used for defining data objects in a compact, human-readable format. A REST API is commonly used for transferring data between server and client, and that data is transferred as JSON.

6.1 JSON compared to XML

If you are already familiar with XML, let's see how JSON and XML look when we store the records of four students in a text-based format.

JSON style:

```
{"students":[
   {"name":"John", "age":"23", "city":"Agra"},
   {"name":"Steve", "age":"28", "city":"Delhi"},
```

[1] Sourced from https://beginnersbook.com/2015/04/json-tutorial/.

```
    {"name":"Peter", "age":"32", "city":"Chennai"},
    {"name":"Chaitanya", "age":"28", "city":"Bangalore"}
]}
```

XML style:

```
<students>
  <student>
    <name>John</name> <age>23</age> <city>Agra</city>
  </student>
  <student>
    <name>Steve</name> <age>28</age> <city>Delhi</city>
  </student>
  <student>
    <name>Peter</name> <age>32</age> <city>Chennai</city>
  </student>
  <student>
    <name>Chaitanya</name> <age>28</age> <city>Bangalore</city>
  </student>
</students>
```

Here you can see JSON is more light-weight compared to XML. Also, in JSON you can take advantage of arrays not available in XML.

7 JSON Data Structure Types and JSONPath

JSON can return lists as an array. In addition, you can reference an element of JSON data using a syntax called "JSON Path".

7.1 JSONPath

JSONPath uses dot and array notation to specify what data element the software should access within a JSON object.

JSONPath syntax can be found at https://goessner.net/articles/JsonPath/index.html.

A helpful tool for testing your JSON Path syntax is https://jsonpath.com/. When specifying your JSONPath in KS, you do not need the "$." root prefix.

7.2 Singular JSON Object

The most basic example of JSON is the singular object.

- Curly brackets start and end the object
- Colons separate name and value components
- Commas separate name and value pairs

This example shows a singular object with three elements:

```
{
  "name" : "Chaitanya Singh",
  "age" : "28",
  "website" : "beginnersbook"
}
```

Inside of an object you can have any number of key-value pairs like we have above. If this object is referred to as "person", you will reference information in a JSON object like

- `person.name` would return Chaitanya Singh.
- `person.age` would return 28
- `person.website` would return beginnersbook

7.3 JSON Objects in an Array

In the above example we stored the information of one person in a JSON object. Now suppose you want to store the information of more than one person - in that case you can have an array of objects. Note the array is enclosed in square brackets "[]" and the elements are separated with commas.

```
[{
   "name" : "Steve",
   "age" :  "29",
   "gender" : "male"
},
{
   "name" : "Peter",
   "age" : "32",
   "gender" : "male"
},
{
   "name" : "Sophie",
   "age" : "27",
   "gender" : "female"
}]
```

If this object is referred to as "students", you could access the information out of this array using the following JSONPath:

- `students[0].age` would return 29
- `students[2].name` would return Sophie
- `students[*].age` would return 29, 32, and 27 in a list or array

7.4 Nesting of JSON Objects

Another way of doing the same thing that was done above uses individually-named nested elements.

```
{
"steve" : {
  "name" : "Steve",
  "age" :  "29",
  "gender" : "male"
},
"pete" : {
  "name" : "Peter",
  "age" : "32",
  "gender" : "male"
},
"sop" : {
  "name" : "Sophie",
  "age" : "27",
  "gender" : "female"
}
}
```

If this object is referred to as "students", you could access the information out of this array using the following JSONPath:

- `students.steve.age` would return 29
- `students.sop.gender` would return female
- `students[0].age` would return an error

8 REST APIs[2]

Representational State Transfer (REST) web services provide interoperability between computer systems on the Internet.

Communications between a client and REST API are done using the same protocols you use when access a web page. Just as a web browser sends a GET command to an HTTP website address and receives an HTML response, a REST API client sends a GET command to an HTTP website address and receives an JSON response.

Although the REST APIs can provide additional formats and protocols, KS's REST API interface is designed to communicate via HTTP and receive JSON responses.

8.1 HTTP URIs and Endpoints

REST APIs are typically addressed using a concatenated base address and endpoint address. For example:

- Base address: `http://www.appdomain.com`
- Endpoint address: `/users/123`
- Parameters (optional): `?size=20&page=5`

These would be combined to a full address: `http://www.appdomain.com/users/123?size=20&page=5`

Endpoints are typically formatted as shown below, although this is not an official REST API requirement and you should therefore check the API documentation to verify the formatting. The following shows how you might interact with an endpoint for animals:

- Retrieving a list of animals: `http://www.samplezoo.com/api/animals`
- Retrieving a list of animals with 20 items from page 5 of the list: `http://www.samplezoo.com/api/animals?size=20&page=5`
- Retrieving, updating, or deleting an animal with id 8844: `http://www.samplezoo.com/api/animals/8844`

8.2 HTTP Response Codes

These are the most common responses you'll see from REST APIs:

HTTP Response Code	Description
200 (Ok)	The server found the resource.
404 (NOT FOUND)	The server did not find the resource.
400 (BAD REQUEST)	The server determined the GET request is incorrectly formed

8.3 HTTP Requests and Response Formats

Calls to a REST API use standard HTTP requests and HTTP responses. This generic message format consists of the following four items.

Item	Examples
Start line	`GET /hello.htm HTTP/1.1 (This is a request line sent by the client)` `HTTP/1.1 200 OK (This is status line sent by the server)`

Zero or more header fields followed by CRLF	`User-Agent: curl/7.16.3 libcurl/7.16.3 OpenSSL/0.9.7l zlib/1.2.3`
	`Host: www.example.com`
	`Accept-Language: en, mi`
	`Date: Mon, 27 Jul 2009 12:28:53 GMT`
	`Server: Apache`
	`Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT`

[2] This section provides are brief introduction to REST APIs. A more extensive introduction can be found at https://en.wikipedia.org/wiki/Representational_state_transfer. Some items from this section were also sourced from this page.

	ETag: "34aa387-d-1568eb00" Accept-Ranges: bytes Content-Length: 51 Vary: Accept-Encoding Content-Type: text/plain
An empty line (i.e., a line with nothing preceding the CRLF) indicating the end of the header fields	-
Message body (optional)	{"name":"Chaitanya Singh","age":"28","website":"beginnersbook"}

8.4 HTTP Verbs

The following table shows the common HTTP verbs typically used in a REST API:

Verb	Description
GET	*Retrieve* the requested resources as specified in the request parameter or body. The GET method is safe, meaning that applying it to a resource does not result in a state change of the resource.
POST	*Create* a resource using the data in the request body.
PUT	*Replace* the resource using the data in the request body or *create* the resource if it does not exist.
DELETE	*Delete* the resource as specified in the request parameter or body.

8.5 Authentication

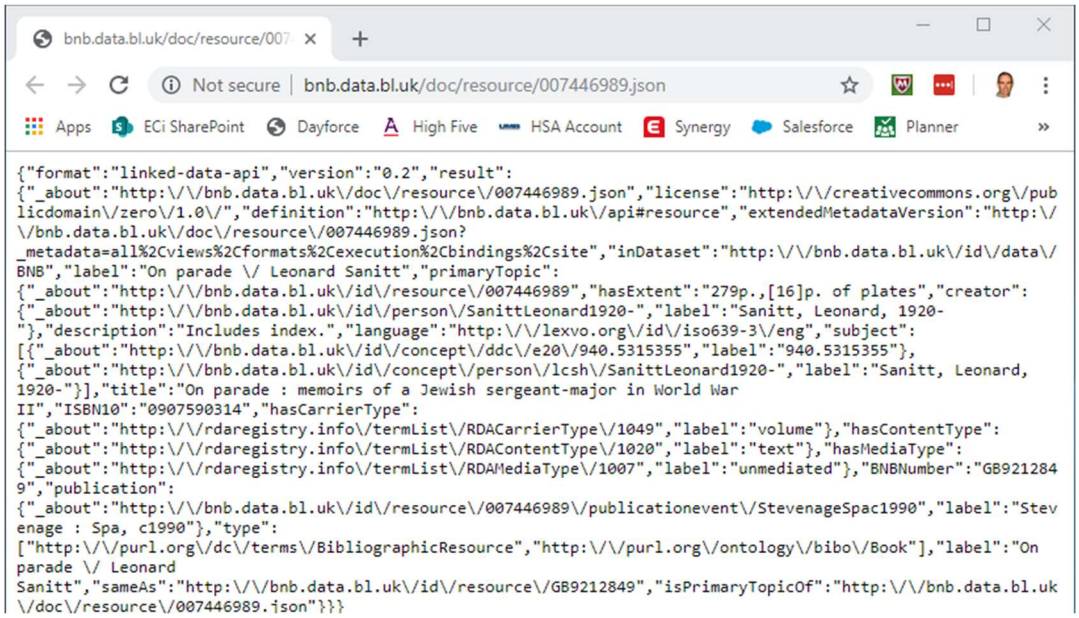
The following is a short summary of REST API authentication methods:

Method	Description
None	The API does not require authentication.
HTTP Basic Authentication	Simply provide a username and password and the API will validate these credentials on every API call.
API Key	You register to use the API, and then the API assigns you a unique API key to identify you when access the API.
OAuth	You register to use the API. Once registered, you use your credentials to request a unique token from the API. The unique token functions much like an API key but will expire after a predetermined time and may limit your API permissions. When your token expires, you use your credentials to request a unique token from the API again.

8.6 Example: GET with Your Web Browser

The following sequence shows a REST API call using your web browser:

1. Open a web browser such as Chrome and enter the following URL: http://bnb.data.bl.uk/doc/resource/007446989.json. This API does not require authentication.
2. You will see a response from the API containing a JSON response:



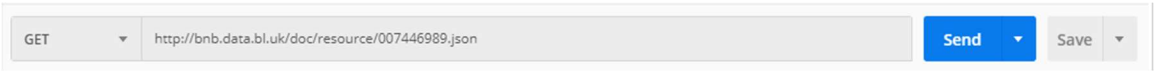
{"format":"linked-data-api","version":"0.2","result":
{"_about":"http:\/\/bnb.data.bl.uk\/doc\/resource\/007446989.json","license":"http:\/\/creativecommons.org\/publicdomain\/zero\/1.0\/","definition":"http:\/\/bnb.data.bl.uk\/api#resource","extendedMetadataVersion":"http:\/\/bnb.data.bl.uk\/doc\/resource\/007446989.json?_metadata=all%2Cviews%2Cformats%2Cexecution%2Cbindings%2Csite","inDataset":"http:\/\/bnb.data.bl.uk\/id\/data\/BNB","label":"On parade \/ Leonard Sanitt","primaryTopic":
{"_about":"http:\/\/bnb.data.bl.uk\/id\/resource\/007446989","hasExtent":"279p.,[16]p. of plates","creator":
{"_about":"http:\/\/bnb.data.bl.uk\/id\/person\/SanittLeonard1920-","label":"Sanitt, Leonard, 1920-"},"description":"Includes index.","language":"http:\/\/lexvo.org\/id\/iso639-3\/eng","subject":
[{"_about":"http:\/\/bnb.data.bl.uk\/id\/concept\/ddc\/e20\/940.5315355","label":"940.5315355"},
{"_about":"http:\/\/bnb.data.bl.uk\/id\/concept\/person\/lcsh\/SanittLeonard1920-","label":"Sanitt, Leonard, 1920-"}],"title":"On parade : memoirs of a Jewish sergeant-major in World War II","ISBN10":"0907590314","hasCarrierType":
{"_about":"http:\/\/rdaregistry.info\/termList\/RDACarrierType\/1049","label":"volume"},"hasContentType":
{"_about":"http:\/\/rdaregistry.info\/termList\/RDAContentType\/1020","label":"text"},"hasMediaType":
{"_about":"http:\/\/rdaregistry.info\/termList\/RDAMediaType\/1007","label":"unmediated"},"BNBNumber":"GB9212849","publication":
{"_about":"http:\/\/bnb.data.bl.uk\/id\/resource\/007446989\/publicationevent\/StevenageSpac1990","label":"Stevenage : Spa, c1990"},"type":
["http:\/\/purl.org\/dc\/terms\/BibliographicResource","http:\/\/purl.org\/ontology\/bibo\/Book"],"label":"On parade \/ Leonard Sanitt","sameAs":"http:\/\/bnb.data.bl.uk\/id\/resource\/GB9212849","isPrimaryTopicOf":"http:\/\/bnb.data.bl.uk\/doc\/resource\/007446989.json"}}}

8.7 Example: Get with Postman

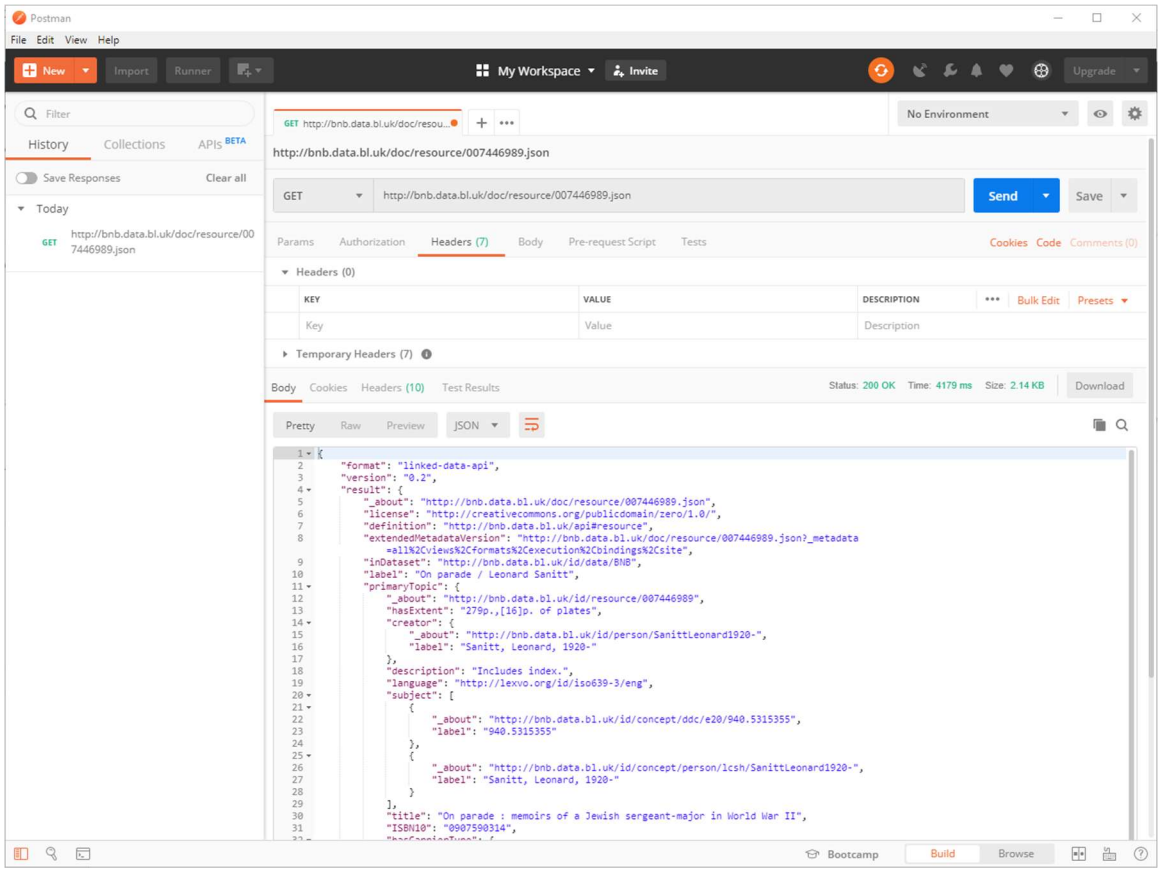
Postman is an excellent tool for troubleshooting REST API calls and responses. You can download and learn about this tool at https://www.getpostman.com/downloads/. The free version will suffice for nearly all needs related to KS.

1. Open Postman. If not already open by default, open a new tab using the plus in the tab bar.

2. Choose the verb on the left. In this example we use the default "GET". Then enter the API URI (call the same API as the previous example). Click Send.



3. Below you see the full screen showing the result body. You'll note this is prettified (JSON standard formatting applied) by default. You can also examine



More information about Postman can be found on the www.getpostman.com website.